

Java Class Example

```
1  /**
2     * Example.java
3     * @author Markus Knauer
4     * @version 1.0
5     */
6  package com.mknauer.example;
7  import java.lang.*;           // this is the default
8  /** This class is an example */
9  public class Example{
10     /** example variable */
11     public int x;
12     /** example method
13         * @param input This is an input parameter
14         * @return Return value is the input value
15         */
16     public int example_method( int input ){
17         return input;
18     }
19 }
```

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example`

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with
`javadoc -author -version Example.java.`

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with
`javadoc -author -version Example.java`.
- The structure of the file `Example.java`

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with
`javadoc -author -version Example.java`.
- The structure of the file `Example.java`
 1. It starts with some comments including a description of the class.

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with

```
javadoc -author -version Example.java.
```
- The structure of the file `Example.java`
 1. It starts with some comments including a description of the class.
 2. Next line is the `package`-statement

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with

```
javadoc -author -version Example.java.
```
- The structure of the file `Example.java`
 1. It starts with some comments including a description of the class.
 2. Next line is the `package`-statement
 3. Followed by one ore more `import`-statements

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with
`javadoc -author -version Example.java`.
- The structure of the file `Example.java`
 1. It starts with some comments including a description of the class.
 2. Next line is the `package-statement`
 3. Followed by one ore more `import-statements`
 4. Then there is exactly *one* `public class`.

Java Class Example – Explanation

- The line `package com.mknauer.example` makes the `Example.class` a member of the package `example` and `javac` produces the necessary directory structure if compiled with option `-d`.
- With the `jar`-tool you can pack all your `.class`-files of your packages into one file. The syntax is similar to the Unix `tar` command. (These archives can be signed for security reasons.)
- In the source file there are different `@tags`. You can compile an html-documentation with
`javadoc -author -version Example.java`.
- The structure of the file `Example.java`
 1. It starts with some comments including a description of the class.
 2. Next line is the `package-statement`
 3. Followed by one or more `import-statements`
 4. Then there is exactly *one* `public class`.
 5. In the class there are declarations of variables and methods.

Wrapper Classes

Wrapper Classes

For every primitive data type there exists a corresponding class type with the same or a similar name. Further information can be found in the package `java.lang`.

<code>byte</code>	<code>java.lang.Byte</code>
<code>short</code>	<code>java.lang.Short</code>
<code>...</code>	<code>...</code>
<code>int</code>	<code>java.lang.Integer</code>
<code>char</code>	<code>java.lang.Character</code>

Wrapper Classes

For every primitive data type there exists a corresponding class type with the same or a similar name. Further information can be found in the package `java.lang`.

<code>byte</code>	<code>java.lang.Byte</code>
<code>short</code>	<code>java.lang.Short</code>
<code>...</code>	<code>...</code>
<code>int</code>	<code>java.lang.Integer</code>
<code>char</code>	<code>java.lang.Character</code>

- Most of their methods are `static` which means you can work directly with the class methods without building objects.

Wrapper Classes

For every primitive data type there exists a corresponding class type with the same or a similar name. Further information can be found in the package `java.lang`.

<code>byte</code>	<code>java.lang.Byte</code>
<code>short</code>	<code>java.lang.Short</code>
<code>...</code>	<code>...</code>
<code>int</code>	<code>java.lang.Integer</code>
<code>char</code>	<code>java.lang.Character</code>

- Most of their methods are `static` which means you can work directly with the class methods without building objects.
- Some examples of class `Character`: `isUpperCase(char)`, `toUpperCase(char)`, `isDigit(char)`, `isLetter(char)`

Wrapper Classes

For every primitive data type there exists a corresponding class type with the same or a similar name. Further information can be found in the package `java.lang`.

<code>byte</code>	<code>java.lang.Byte</code>
<code>short</code>	<code>java.lang.Short</code>
<code>...</code>	<code>...</code>
<code>int</code>	<code>java.lang.Integer</code>
<code>char</code>	<code>java.lang.Character</code>

- Most of their methods are `static` which means you can work directly with the class methods without building objects.
- Some examples of class `Character`: `isUpperCase(char)`, `toUpperCase(char)`, `isDigit(char)`, `isLetter(char)`
- Some examples of class `Integer`: `MAX_VALUE`, `parseInt(String)`, `toString()`, ...

Wrapper Class Example

Wrapper Class Example

```
1 String s = "12";
2 int i = Integer.parseInt( s );
3
4 System.out.println ( Integer.MAX_VALUE );
5
6 char ch = 'a';
7
8 if ( Character.isDigit ( ch ) )
9     System.out.println ( "'a' is a digit ." );
10
11 if ( Character.isLetter ( ch ) )
12     System.out.println ( "'a' is a letter ." );
```

Vector **Class**

Vector Class

The `Vector` class (`java.util`) implements a growable array of objects. It contains components that can be accessed using an integer index. The size can grow or shrink after the vector has been created.

```
1  import java.util .*;    // line at the top of your file
2
3  Vector v = new Vector();
4  String s1 = "string_one";
5
6  v.addElement( s1 );
7  v.addElement( new String( "string_two" ) );
8
9  for( int i=0; i<v.size (); i++ )
10     System.out.println ( ( String)v.elementAt( i ) );
```

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement()).intValue ();
```

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement()).intValue ();
```

Some methods of the `Vector` class:

`addElement(Object o)` Adds the specified component to the end of the vector, increasing the size by one.

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement()).intValue ();
```

Some methods of the `Vector` class:

`addElement(Object o)` Adds the specified component to the end of the vector, increasing the size by one.

`clear()` Removes all elements from the vector.

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement()).intValue ();
```

Some methods of the `Vector` class:

`addElement(Object o)` Adds the specified component to the end of the vector, increasing the size by one.

`clear()` Removes all elements from the vector.

`elementAt(int index)` Returns the component at the specified index.

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement()).intValue ();
```

Some methods of the `Vector` class:

`addElement(Object o)` Adds the specified component to the end of the vector, increasing the size by one.

`clear()` Removes all elements from the vector.

`elementAt(int index)` Returns the component at the specified index.

`size()` Returns the number of components in this vector.

Vector Class

Important: You store *references to objects* in the `Vector`! – **Typecast!**

```
1 v.addElement( new Integer( 23 ) );  
2 int i = (( Integer)v.lastElement ()).intValue ();
```

Some methods of the `Vector` class:

`addElement(Object o)` Adds the specified component to the end of the vector, increasing the size by one.

`clear()` Removes all elements from the vector.

`elementAt(int index)` Returns the component at the specified index.

`size()` Returns the number of components in this vector.

`remove(int index)` Removes the element at the specified position in this vector.

StringBuffer Class

The main difference between `String` and `StringBuffer`: While an object of type `String` is static a `StringBuffer` object can be modified.

StringBuffer Class

The main difference between `String` and `StringBuffer`: While an object of type `String` is static a `StringBuffer` object can be modified.

```
1  String sTestString = "ABCGHI";
2  StringBuffer sTestSB = new StringBuffer( sTestString );
3
4  sTestSB.append( "JKL" );
5  sTestSB.insert ( 3, "DEF" );
6
7  System.out.println ( sTestSB );
8
9  sTestSB.setCharAt( 0, 'a' );
10 sTestSB.reverse();
11
12 sTestString = sTestSB.toString();
13 System.out.println ( sTestString );
```

Some Mathematic Functions

The `Math` class (`java.lang`) contains lots of important static class methods and constants.

Some Mathematic Functions

The `Math` class (`java.lang`) contains lots of important static class methods and constants.

A few examples:

```
1 System.out.println ( Math.PI );
2 System.out.println ( Math.E );
3
4 System.out.println ( "Cosinus_of_0.5:" + Math.cos ( 0.5 ) );
5 System.out.println ( "5_to_the_power_of_3:" + Math.pow ( 5, 3 ) );
6 System.out.println ( "Random_number_between_0_and_1:" + Math.random() );
```

HashMap Class

This class is a hash table based implementation of the `Map` interface.
(Package `java.util`)

```
1 import java.util.*;
2
3 public class HashMap01 {
4     public static void main( String [] args ) {
5
6         HashMap hm = new HashMap();
7
8         for( int i=0; i<args.length; i++ ) {
9             hm.put( new Integer( i ), args[i ] );
10        }
11
12        System.out.println ( hm );
13        System.out.println ( hm.keySet() );
14    }
15 }
```

First Introduction to Exceptions

```
1  int x = 5;  
2  int y = 0;  
3  
4  y = x / 0;    // division by zero!
```

This code contains a runtime error and the program will stop here.

First Introduction to Exceptions

```
1  int x = 5;  
2  int y = 0;  
3  
4  y = x / 0;    // division by zero!
```

This code contains a runtime error and the program will stop here.

Solution: Programming with exceptions:

First Introduction to Exceptions

```
1  int x = 5;
2  int y = 0;
3
4  y = x / 0;    // division by zero!
```

This code contains a runtime error and the program will stop here.

Solution: Programming with exceptions:

```
1  ...
2  try {
3      y = x / 0;
4  }
5  catch( Exception e ) {
6      System.out.println ( "Error with divison" );
7      e.printStackTrace ();
8  }
```

File Handling

The classes for file handling and manipulating are placed in the package `java.io`.

File Handling

The classes for file handling and manipulating are placed in the package `java.io`.

`File` is an abstract representation of file and directory pathnames.

File Handling

The classes for file handling and manipulating are placed in the package `java.io`.

`File` is an abstract representation of file and directory pathnames.

`FileReader` **and** `FileWriter` are meant for reading and writing streams of *character*.

File Handling

The classes for file handling and manipulating are placed in the package `java.io`.

`File` is an abstract representation of file and directory pathnames.

`FileReader` **and** `FileWriter` are meant for reading and writing streams of *character*.

`BufferedReader` **and** `BufferedWriter` reads/writes from a character-input/output stream, buffering characters so as to provide for the efficient reading/writing of characters, arrays, and lines.

File Class Example

```
1 File fTestFile = new File( " File . txt " );
2
3 if ( fTestFile . exists ( ) )
4     System.out.println ( " File . txt _ exists " );
5 if ( fTestFile . isFile ( ) )
6     System.out.println ( " File . txt _ is _ a _ regular _ file " );
7 if ( fTestFile . isDirectory ( ) )
8     System.out.println ( " File . txt _ is _ a _ directory " );
9
10 System.out.println ( " Absolute _ pathname _ of _ file : _ "
11                     + fTestFile . getAbsolutePath ( ) );
12
13 System.out.println ( " System _ dependend _ separator _ is _ "
14                     + File . separator );
```

Reading Character Files

```
1 File fTestFile = new File( " file .txt" );
2 try {
3     FileReader frTestFile = new FileReader( fTestFile );
4     int c;
5     while ( ( c=frTestFile .read () ) != -1) {
6         // do something i.e. print the content of c
7         System.out.print ( ( char)c );
8     }
9     frTestFile .close ();
10 }
11 catch ( IOException e ) {
12     e.printStackTrace ();
13 }
```

Reading and Writing Character Files with Buffer

```
1  try {
2      BufferedReader br = new
3          BufferedReader( new FileReader( "TestIn.txt" ) );
4      BufferedWriter bw = new
5          BufferedWriter( new FileWriter( "TestOut.txt" ) );
6
7      String sLine;
8
9      while ( ( sLine=br.readLine () ) != null ) {
10         bw.write( sLine );
11         bw.newLine();
12     }
13
14     br.close ();
15     bw.close ();
16 }
17 catch ( IOException e ) {
18     e.printStackTrace ();
19 }
```