CHAPTER 18

JAVASCRIPT / NODE.JS PROGRAMMING

On the following pages you'll find some simple JavaScript/Node.js programs explained.

You should print these pages double-sided.

2017-10-15 File created.
2017-11-27 Some object-oriented programs explained.

Copyright © Kari Laitinen

Game.js – A simple program

This line is not actually part of the program. This is a comment line that gives documentary information to the reader of the program. A double slash // marks the beginning of a comment. The compiler or program interpreter ignores the double slash and the text that follows it on the same line.

```
//  Game.js (c) Kari Laitinen

process.stdout.write( "\n This program is a computer game. Please, "
                    + "\n type in a number : " ) ;

process.stdin.on( "data", function( input_from_user )
{
   // The statements below will be executed automatically after
   // the user has typed in a number.

   var given_number = Number( input_from_user ) ;

   var winning_number = given_number + 1 ;

   process.stdout.write( "\n You typed in " + given_number + "."
                      +  "\n My number is " + winning_number  + "."
                      +  "\n Sorry, you lost. I won. The game is over.\n\n" ) ;

   process.exit() ;

} ) ;
```

Texts inside double quote characters " " are strings of characters that will be displayed on the screen. **\n** among the text means that the text will begin from a new line. **\n** is said to be the newline character.

Game.js - 1.+ A program that implements a simple computer game.

When we write **process.stdout**, we can refer to the screen of the computer. **write()** is a method that can be invoked by writing **process.stdout.write(...)**, and this method invocation (or method call) writes data to the screen.

```
var given_number = Number( input_from_user ) ;

var winning_number = given_number + 1 ;

process.stdout.write( "\n You typed in " + given_number + "."
                    + "\n My number is " + winning_number  + "."
                    + "\n Sorry, you lost. I won. The game is over.\n\n" ) ;
```

It is possible to output many types of data in a single call to method process.stdout.write(). Here the values of the variables are displayed between strings of characters given inside double quotes. Operator + is placed between different types of data. The + operator converts the numerical values stored in the variables to character strings, and joins these character strings to the other character strings given inside double quotes. A semicolon (;) terminates the entire statement.

Game.js - 1 - 1. Part of the program explained.

```
H:\jsprograms\nodejsfiles2>node Game.js

 This program is a computer game. Please,
 type in a number : 1234

 You typed in 1234.
 My number is 1235.
 Sorry, you lost. I won. The game is over.


H:\jsprograms\nodejsfiles2>node Game.js

 This program is a computer game. Please,
 type in a number : 667788

 You typed in 667788.
 My number is 667789.
 Sorry, you lost. I won. The game is over.
```

Game.js - X. Here the program is executed twice.

Likejavascript.js – An if construct demonstrated

In JavaScript either single quotes or double quotes can be used to write string literals. Thus, 'Y' and "Y" have the same meaning. When dealing with single-character strings, I use single quotes.

```
//  Likejavascript.js (c) Kari Laitinen

process.stdout.write( "\n Do you like the JavaScript language?"
                    +  "\n Please, answer Y or N :   " ) ;

process.stdin.on( "data", function( input_from_user )
{
   // The following statements will be executed after the user
   // of this program has typed in a response.

   // given_letter will refer to the first letter
   // of the user response.

   var given_letter = ( String( input_from_user ) )[ 0 ] ;

   if ( given_letter  ==  'Y' ||  given_letter  ==  'y' )
   {
      process.stdout.write( "\n That's nice to hear. \n\n" ) ;
   }
   else if ( given_letter  ==  'N' || given_letter  ==  'n' )
   {
      process.stdout.write( "\n That is not so nice to hear. "
                         +  "\n I hope you change your mind soon.\n\n" ) ;
   }
   else
   {
      process.stdout.write( "\n I do not understand \""
                         +  given_letter   +  "\".\n\n" ) ;
   }

   process.exit() ;  // This terminates the program.
} ) ;
```

This statement will be executed if `given_letter` contains something other than 'Y', 'y', 'N', or 'n'. Note that if you want to include a double quote character (") among the text to be printed, you must write a backslash \ before it.

This boolean expression is true if `given_letter` contains either the character of uppercase N or lowercase n. || is the logical-OR operator which can combine relational expressions.

Likejavascript.js - 1.+ A program containing an if-else if-else construct.

This logical-OR operator || combines the two relational expressions into one boolean expression, which is true when at least one of the relational expressions is true. The entire boolean expression is false only when both relational expressions are false.

```
if ( given_letter  ==  'Y' ||  given_letter  ==  'y' )
```

Likejavascript.js - 1-1. The first boolean expression in the program.

```
H:\jsprograms\nodejsfiles2>node Likejavascript.js

 Do you like the JavaScript language?
 Please, answer Y or N :  y

 That's nice to hear.

H:\jsprograms\nodejsfiles2>node Likejavascript.js

 Do you like the JavaScript language?
 Please, answer Y or N :  z

 I do not understand "z".
```

Likejavascript.js - X. The program is executed here twice, with inputs y and z.

ForLoopTraditional.js – C-style for loop

Inside the parentheses after the keyword **for**, **for** loops have three "things" separated with two semicolons. In this loop

- the assignment statement `number_to_print = 0` will be executed before the program actually starts looping,
- the boolean expression `number_to_print <= 20` decides when the loop terminates, and
- the increment statement `number_to_print ++` will be executed each time after the internal statement of the loop has been executed.

```
//  ForLoopTraditional.js (c) Kari Laitinen

process.stdout.write( "\n Numbers from 0 to 20 are the following:\n\n ") ;

for ( let number_to_print  =  0  ;
          number_to_print  <= 20 ;
          number_to_print  ++  )
{
   process.stdout.write( " "  +  number_to_print ) ;
}

process.stdout.write( "\n\n End of program.\n\n ") ;
```

In the same way as in the case of **while** loops, the internal statements of **for** loops are written inside braces. The internal statements of a loop can also be called with the term "body of the loop".

This statement is the only statement inside the loop. The statement will be executed 21 times. When the value of **number_to_print** is 20, it will be incremented to 21, resulting in that the boolean expression
`number_to_print <= 20`
is not true any more, and the loop terminates.

ForLoopTraditional.js - 1. Program WhileLoop.js implemented with a for loop.

```
H:\jsprograms\nodejsfiles2>node ForLoopTraditional.js

 Numbers from 0 to 20 are the following:

  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

 End of program.
```

ForLoopTraditional.js - X. The output of the program is always the same.

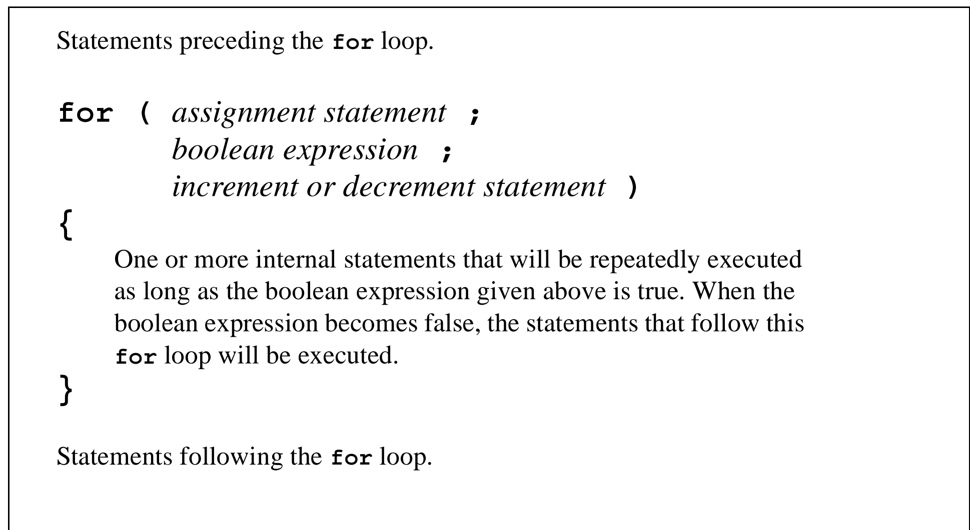

Figure 6-6. Typical structure of a for loop.

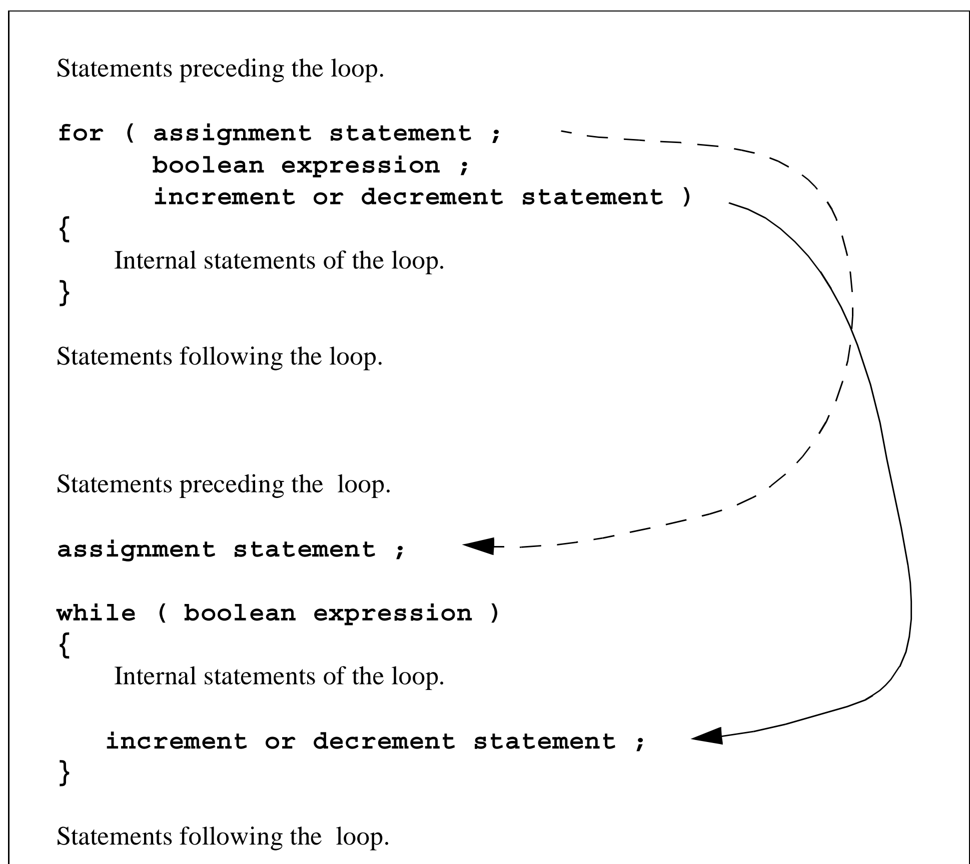

Figure 6-7. Converting a for loop into a while loop

Reverse.js – storing numbers into an array

```
//  Reverse.js (c) Kari Laitinen

var given_numbers = [] ;

process.stdout.write( "\n This program reads numbers from the keyboard."
                   +  "\n After receiving a zero, it prints the numbers"
                   +  "\n in reverse order. Please, start entering numbers."
                   +  "\n The program will stop when you enter a zero.\n\n" ) ;

process.stdout.write( "   "  +  given_numbers.length
                   +  "  Enter a number: ") ;

process.stdin.on( "data", function( input_from_user )
{
   var number_from_keyboard = Number( input_from_user ) ;

   // The push() method adds the number to the end of the array.

   given_numbers.push( number_from_keyboard ) ;

   if ( number_from_keyboard == 0 )
   {
      process.stdout.write( "\n Reverse order:  " ) ;

      var number_index = given_numbers.length ;

      while ( number_index > 0 )
      {
         number_index -- ;
         process.stdout.write( given_numbers[ number_index ] + "    " ) ;
      }

      process.stdout.write( "\n\n" ) ;

      process.exit() ;  // This terminates the program.
   }
   else
   {
      // We are not finished. Let's ask for one more number.
      // The length property tells how many elements the array has,
      // i.e. how many numbers have been entered.

      process.stdout.write( "   "  +  given_numbers.length
                         +  "  Enter a number: ") ;
   }
} ) ;
```

This `while` loop prints the contents of the array to the screen. Because `number_index` is decremented inside the loop, the numbers are printed in reverse order.

Reverse.js - 1.+ A program that inputs numbers and prints them in reverse order.

When this loop is entered, **number_index** contains a value that equals the number of numbers that were originally typed in and stored to the array. **number_index** must be decremented before anything is printed. Operator -- decrements the value of a variable by one.

This loop can be approximately translated into natural language: "As long as it is possible to decrement variable **number_index** without it becoming negative, decrement it, and use the value to print a number from the array."

```
while ( number_index > 0 )
{
   number_index -- ;
   process.stdout.write( given_numbers[ number_index ] + "   " ) ;
}
```

Three spaces are used to separate the printed numbers.

Reverse.js - 1 - 1. The while loop that prints the array in reverse order.

```
H:\jsprograms\nodejsfiles2>node Reverse.js

 This program reads numbers from the keyboard.
 After receiving a zero, it prints the numbers
 in reverse order. Please, start entering numbers.
 The program will stop when you enter a zero.

   0  Enter a number: 22
   1  Enter a number: 33
   2  Enter a number: 444
   3  Enter a number: 555
   4  Enter a number: 6666
   5  Enter a number: 7777
   6  Enter a number: 88
   7  Enter a number: 99
   8  Enter a number: 0

 Reverse order:  0   99   88   7777   6666   555   444   33   22
```

Reverse.js - X. Here the program is executed with 9 numbers.

StringReverse.js – accessing characters of a string

After this statement has been executed, **string_from_keyboard** points to a string containing the characters that were typed in from the keyboard.

```
//  StringReverse.js (c) Kari Laitinen

process.stdout.write( "\n This program is able to reverse a string."
                   +  "\n Please, type in a string.\n\n   " ) ;

process.stdin.on( "data", function( input_from_user )
{
   // When we take the string from the user input, we need to
   // use the trim() method to get rid of line termination characters.

   var string_from_keyboard = input_from_user.toString().trim() ;

   process.stdout.write( "\n String length is " + string_from_keyboard.length
                     +  ".\n\n String in reverse character order: \n\n   " ) ;

   var character_index = string_from_keyboard.length ;

   while (  character_index  >  0  )
   {
      character_index  --  ;
      process.stdout.write( string_from_keyboard[ character_index ] ) ;
   }

   process.stdout.write( "\n\n" ) ;
   process.exit() ;
} ) ;
```

The property **length** reveals the length of a string, i.e., it returns the number of characters in the string.

Inside the **while** loop, **character_index** is first decremented, and then the character referred to by that index value is printed. This way characters are printed from the end to the beginning of the string. Before this loop is entered, **character_index** has a value that is the first illegal index value.

StringReverse.js - 1. Printing the characters of a string in reverse order.

```
H:\jsprograms\nodejsfiles2>node StringReverse.js

 This program is able to reverse a string.
 Please, type in a string.

   Alan Turing is a famous man in the history of computing.

 String length is 56.

 String in reverse character order:

   .gnitupmoc fo yrotsih eht ni nam suomaf a si gniruT nalA
```

In 1937, before any electronic computers were built, Alan Turing published a mathematical model for a computer.

StringReverse.js - X. Reversing a long input string.

Continents.js – string methods explored

```
//  Continents.js (c) Kari Laitinen

/*  This program shows how some common String methods work.

      - charAt() returns the character in the indexed position
      - an index inside brackets is an alternative to charAt()
      - indexOf() returns the index of the first occurrence of a substring
      - lastIndexOf() retruns the index of the last occurrence of a substring
      - substring() returns the specified substring
      - replace() replaces occurrences of a substring
      - toUpperCase() returns a string in which all letters are converted
        to upper case if necessary.  */

var indexes = "012345678901234567890123456789012345678901234567890123456" ;

var continent_names =
              "America Europe Africa Asia Australia Antarctica" ;

process.stdout.write( "\n   "  +  indexes  ) ;

process.stdout.write( "\n   "  +  continent_names  +  "\n" ) ;

process.stdout.write( "\n   "  +  continent_names.charAt( 8 ) ) ;
process.stdout.write( "\n   "  +  continent_names[ 8 ] ) ;

process.stdout.write( "\n   "  +  continent_names.indexOf( "ia" ) ) ;

process.stdout.write( "\n   "  +  continent_names.lastIndexOf( "ia" ) ) ;

process.stdout.write( "\n   "  +  continent_names.indexOf( "Atlantis" ) ) ;

process.stdout.write( "\n   "  +  continent_names.substring( 15, 26 ) ) ;
process.stdout.write( "\n   "  +  continent_names.substring( 27 ) ) ;

process.stdout.write( "\n   "  +  continent_names.substring( 27 )
                            +  continent_names.substring( 0, 27 ) ) ;

// The following replaces the first occurrence of "ica" with "XXX".
process.stdout.write( "\n   "  +  continent_names.replace( "ica", "XXX" ) ) ;

// The regular expression replaces all occurrences of "ica" with "XXX".
process.stdout.write( "\n   "  +  continent_names.replace( /ica/g, "XXX" ) ) ;

process.stdout.write( "\n   "  +  continent_names.toUpperCase() ) ;

process.stdout.write( "\n\n" ) ;
```

Continents.js - 1. A program that uses the most common string methods.

```
H:\jsprograms\nodejsfiles2>node Continents.js

   01234567890123456789012345678901234567890123456
   America Europe Africa Asia Australia Antarctica

   E
   E
   24
   34
   -1
   Africa Asia
   Australia Antarctica
   Australia AntarcticaAmerica Europe Africa Asia
   AmerXXX Europe Africa Asia Australia Antarctica
   AmerXXX Europe AfrXXX Asia Australia AntarctXXX
   AMERICA EUROPE AFRICA ASIA AUSTRALIA ANTARCTICA
```

-1 means that the string "Atlantis" was not found inside the string that contains names of continents.

Continents.js - X. The program executed.

Rectangles.js – A program that contains a simple class

Class **Rectangle** has three data fields that are referred to with keyword **this**. **this.rectangle_width**, **this.rectangle_height**, and **this.filling_character** are data items that belong to every object of type **Rectangle**. These data items are called fields in programming terminology. Fields are data members of a class.

```
//  Rectangles.js (c) Kari Laitinen

class Rectangle
{
   constructor( given_rectangle_width,
                given_rectangle_height,
                given_filling_character )
   {
      this.rectangle_width    =  given_rectangle_width ;
      this.rectangle_height   =  given_rectangle_height ;
      this.filling_character  =  given_filling_character ;
   }

   print()
   {
      for ( let row_counter  =  0 ;
                row_counter  <  this.rectangle_height ;
                row_counter  ++ )
      {
         process.stdout.write( "\n      " ) ;

         for ( let character_counter  =  0 ;
                   character_counter  <  this.rectangle_width ;
                   character_counter  ++ )
         {
            process.stdout.write( this.filling_character ) ;
         }
      }

      process.stdout.write( "\n" ) ;
   }
} // End of class Rectangle
```

A method of a class can freely read and write the data fields, the classwide data, that are declared with keyword **this**.

In addition to constructor, class **Rectangle** has one method named **print()**. Because the keyword **static** is not used in the declaration of the method, it is a non-static instance method than can only be called in relation to a **Rectangle** object according to the following statement syntax

```
object_reference_name.method_name( ... ) ;
```

Rectangles.js - 1: The declaration of class Rectangle.

This statement declares a variable **first_rectangle**, creates a **Rectangle** object, and makes **first_rectangle** reference the created object. This statement could be replaced with the statements

```
var first_rectangle ;
first_rectangle  =  new  Rectangle( 7, 4, 'Z' ) ;
```

```
// Here is the 'main' program in which two Rectangle
// objects are created. Method print() is called for both objects.

var first_rectangle  =  new Rectangle( 7, 4, 'Z' ) ;

first_rectangle.print() ;

var second_rectangle  =  new Rectangle( 12, 3, 'X' ) ;

second_rectangle.print() ;

process.stdout.write( "\n" ) ;
```

Rectangles.js - 2. The "main" program that creates two Rectangle objects.

```
D:\jsprograms\nodejsfiles3>node Rectangles.js

      ZZZZZZZ
      ZZZZZZZ
      ZZZZZZZ
      ZZZZZZZ

      XXXXXXXXXXXX
      XXXXXXXXXXXX
      XXXXXXXXXXXX
```

This is the **Rectangle** object referenced by **second_rectangle**. The printed rectangle is 12 character positions wide, 3 rows high, and filled with character X.

Rectangles.js - X. The rectangles are made by printing a single character repeatedly.

Animals.js – Animal objects in use

```
//  Animals.js (c) Kari Laitinen

class  Animal
{
   constructor( given_parameter )
   {
      if ( typeof given_parameter == "string" ||
           given_parameter instanceof String )
      {
         //  The given parameter is a string. We suppose that a new
         //  Animal object with the given species name is being constructed.

         this.species_name      =  given_parameter ;
         this.stomach_contents  =  "" ;
      }
      else if ( given_parameter instanceof Animal )
      {
         //  A reference to an Animal object was given as an actual parameter.
         //  The new Animal object will be a copy of the given Animal object.

         this.species_name      =  given_parameter.species_name ;
         this.stomach_contents  =  given_parameter.stomach_contents ;
      }
      else
      {
         process.stdout.write(
            "\n Unacceptable object was given to Animal constructor.\n" ) ;
      }

   }


   feed( food_for_this_animal )
   {
      this.stomach_contents  =
          this.stomach_contents  +  food_for_this_animal  +  ", "  ;
   }

   make_speak()
   {
      process.stdout.write( "\n Hello, I am a " + this.species_name     + "."
                          + "\n I have eaten: " + this.stomach_contents + "\n" ) ;
   }
} // End of class Animal
```

`Animal` objects are fed by concatenating (appending) the food string to previous stomach contents. Operator + joins a new string to the end of an existing string. `stomach_contents` references a new `String` object after this operation.

Animals.js - 1: The declaration of class Animal.

The first two **Animal** objects are created so that a string is given as a parameter for the constructor. The constructor initializes the stomachs of these **Animal** objects with an empty string. Objects are often said to be instances of a class. The word 'instantiate' is used refer to the creation of an object.

```
// Here is the 'main' program in which three Animal
// objects are created.

var cat_object  =  new Animal( "cat" ) ;
var dog_object  =  new Animal( "vegetarian dog" ) ;

cat_object.feed( "fish" ) ;
cat_object.feed( "chicken" ) ;

dog_object.feed( "salad" ) ;
dog_object.feed( "potatoes" ) ;

var another_cat  =  new Animal( cat_object ) ;

another_cat.feed( "milk" ) ;

cat_object.make_speak() ;
dog_object.make_speak() ;
another_cat.make_speak() ;

process.stdout.write( "\n" ) ;
```

When **another_cat** is made to speak here, it is no longer an identical copy of **cat_object** because it was fed with milk after the cloning operation.

The object referenced by **another_cat** becomes a (shallow) copy of the object referenced by **cat_object**.

Animals.js - 2. The "main" program that creates and uses Animal objects.

```
D:\jsprograms\nodejsfiles3>node Animals.js

 Hello, I am a cat.
 I have eaten: fish, chicken,

 Hello, I am a vegetarian dog.
 I have eaten: salad, potatoes,

 Hello, I am a cat.
 I have eaten: fish, chicken, milk,
```

Animals.js - X. All these lines are generated through calls to method make_speak().

BankPolymorphic.js – Redefining methods in derived classes

The constructor of a class is named **constructor**. The compiler generates a call to a constructor when an object is created. Constructors are like methods as they can get parameters. This constructor simply copies the values of its parameters to the fields of the class.

```
class  BankAccount
{
   constructor( given_account_owner,
                given_account_number,
                initial_balance )
   {
      this.account_owner    =  given_account_owner ;
      this.account_number   =  given_account_number ;
      this.account_balance  =  initial_balance  ;
   }

   show_account_data()
   {
      process.stdout.write( "\n\nB A N K   A C C O U N T   D A T A : "
                         +  "\n   Account owner :  "  +  this.account_owner
                         +  "\n   Account number:  "  +  this.account_number
                         +  "\n   Current balance: "  +  this.account_balance ) ;
   }

   deposit_money( amount_to_deposit )
   {
      process.stdout.write( "\n\nTRANSACTION FOR ACCOUNT OF " +  this.account_owner
                         +  " (Account number " +  this.account_number  + ")" ) ;
      process.stdout.write( "\n   Amount deposited: "  +  amount_to_deposit
                         +  "\n   Old account balance: "  +  this.account_balance ) ;
      this.account_balance  =  this.account_balance  +  amount_to_deposit ;
      process.stdout.write( "   New balance: "  +  this.account_balance  ) ;
   }
```

BankPolymorphic.js - 1: A program with several bank account classes.

In JavaScript, instance methods can usually be redefined (overridden) in derived classes Nothing in this method declaration indicates that this particular method is redefined in the other classes of this program. When the compiler finds a method with the same name in a derived class, it knows that this method is overridden in the derived class.

```
   withdraw_money( amount_to_withdraw )
   {
      process.stdout.write( "\n\nTRANSACTION FOR ACCOUNT OF " +  this.account_owner
                          +  " (Account number " +  this.account_number  + ")" ) ;

      if ( this.account_balance  <  amount_to_withdraw )
      {
         process.stdout.write("\n   -- Transaction not completed: "
                + "Not enough money to withdraw " + amount_to_withdraw ) ;
      }
      else
      {
         process.stdout.write("\n   Amount withdrawn:    "  +  amount_to_withdraw
                                + "\n   Old account balance: "  +  this.account_balance )
         this.account_balance  =  this.account_balance  -  amount_to_withdraw ;
         process.stdout.write("   New balance: "  +  this.account_balance ) ;
      }
   }

   transfer_money_to( receiving_account,
                      amount_to_transfer )
   {
      process.stdout.write( "\n\nTRANSACTION FOR ACCOUNT OF " +  this.account_owner
                          +  " (Account number " +  this.account_number  + ")" ) ;

      if ( this.account_balance  >=  amount_to_transfer )
      {
         receiving_account.account_balance  =
                 receiving_account.account_balance  +  amount_to_transfer ;

         process.stdout.write("\n   " + amount_to_transfer + " was transferred to "
              + receiving_account.account_owner  +  " (Account no. "
              + receiving_account.account_number +  ")."
              + "\n   Balance before transfer: " +  this.account_balance ) ;
         this.account_balance  =  this.account_balance  -  amount_to_transfer ;
         process.stdout.write("   New balance:  " +  this.account_balance ) ;
      }
      else
      {
         process.stdout.write( "\n   -- Not enough money for transfer." ) ;
      }
   }
}
```

BankPolymorphic.js - 2: The other part of class BankAccount.

The constructor of superclass **BankAccount** is called here in order to handle the first three constructor parameters. The fourth parameter is copied to the new field of this class.

```
// AccountWithCredit is a subclass of BankAccount.
// Method withdraw_money() is redefined (overridden).

class  AccountWithCredit  extends  BankAccount
{
   constructor( given_account_owner,
                given_account_number,
                initial_balance,
                given_credit_limit  )
   {
      super( given_account_owner,
             given_account_number,
             initial_balance ) ;

      this.credit_limit  =  given_credit_limit ;
   }

   withdraw_money( amount_to_withdraw )
   {
      process.stdout.write( "\n\nTRANSACTION FOR ACCOUNT OF " +  this.account_owner
                         +  " (Account number " +  this.account_number  + ")" ) ;

      if ( this.account_balance  +  this.credit_limit  <  amount_to_withdraw )
      {
         process.stdout.write(
               "\n   --  Transaction not completed: "
             + "Not enough credit to withdraw "+ amount_to_withdraw ) ;
      }
      else
      {
         process.stdout.write(
                "\n   Amount withdrawn:    "  +  amount_to_withdraw
              + "\n   Old account balance: "  +  this.account_balance ) ;
         this.account_balance  =  this.account_balance  -  amount_to_withdraw ;
         process.stdout.write( "   New balance: "  +  this.account_balance ) ;
      }
   }
}
```

The method above overrides the **withdraw_money()** method of the superclass **BankAccount**. All other methods are inherited from the **BankAccount** class in the usual way.

BankPolymorphic.js - 3: Class AccountWithCredit and its redefined withdraw_money().

This is another class that has been derived from class **BankAccount** so that method **withdraw_money()** is redefined. The bank account objects of class **RestrictedAccount** do not allow withdrawals that exceed the specified withdrawal limit. This restriction ensures that the money that is in the account is not spent too quickly.

```
class  RestrictedAccount  extends  BankAccount
{
   constructor( given_account_owner,
                given_account_number,
                initial_balance,
                given_withdrawal_limit  )
   {
      super( given_account_owner,
             given_account_number,
             initial_balance ) ;

      this.maximum_amount_to_withdraw  =  given_withdrawal_limit ;
   }

   withdraw_money( amount_to_withdraw )
   {
      if ( amount_to_withdraw  >  this.maximum_amount_to_withdraw )
      {
         process.stdout.write(
                "\n\nTRANSACTION FOR ACCOUNT OF " +  this.account_owner
             +  " (Account number " +  this.account_number  + ")" ) ;

         process.stdout.write(
                "\n   -- Transaction not completed: Cannot withdraw "
             +  amount_to_withdraw   + "\n   -- Withdrawal limit is "
             +  this.maximum_amount_to_withdraw + "." ) ;
      }
      else
      {
         super.withdraw_money( amount_to_withdraw ) ;
      }
   }
}
```

In the case that the withdrawal is acceptable, when the amount to be withdrawn is not too large, the **withdraw_money()** method of superclass **BankAccount** is called to perform the actual withdrawal. With the help of the reserved keyword **super** it is possible to call the same method in the superclass.

BankPolymorphic.js - 4: Class RestrictedAccount and its version of withdraw_money().

`stones_account` references an object of type `AccountWithCredit`. It is possible to withdraw 2500.00 from this account because the credit limit allows the account balance to go negative.

```
// Here is the 'main' program:

var beatles_account =  new  BankAccount( "John Lennon", 222222, 2000.00 ) ;

var stones_account  =  new  AccountWithCredit( "Brian Jones", 333333,
                                               2000.00,  1000.00 ) ;

var doors_account   =   new  RestrictedAccount( "Jim Morrison", 444444,
                                               4000.00,  1000.00 ) ;

beatles_account.withdraw_money( 2500.00 ) ;
stones_account.withdraw_money( 2500.00 ) ;
doors_account.withdraw_money( 2500.00 ) ;

process.stdout.write( "\n\n" ) ;
```

Here method `withdraw_money()` is called for each created bank account object. Because `withdraw_money()` is a polymorphic method that has several versions of itself, a different method is executed in each case.

There is a withdrawal limit of 1000.00 in the object referenced by `doors_account`. Although the account's initial balance is 4000.00, it is not possible to withdraw 2500.00 because that exceeds the withdrawal limit.

BankPolymorphic.js - 5. Withdrawing money from different bank accounts.

```
D:\jsprograms\nodejsfiles3>node BankPolymorphic.js


TRANSACTION FOR ACCOUNT OF John Lennon (Account number 222222)
   -- Transaction not completed: Not enough money to withdraw 2500

TRANSACTION FOR ACCOUNT OF Brian Jones (Account number 333333)
   Amount withdrawn:    2500
   Old account balance: 2000   New balance: -500

TRANSACTION FOR ACCOUNT OF Jim Morrison (Account number 444444)
   -- Transaction not completed: Cannot withdraw 2500
   -- Withdrawal limit is 1000.
```

BankPolymorphic.js - X. The withdrawal succeeds only for stones_account.

The class hierarchy of **BankPolymorphic.js** can be described as shown in Figure 12-4. The two new classes that are derived from class `BankAccount` in **BankPolymorphic.js** can be described in the following way

- Class `AccountWithCredit` allows the balance of an account object to be negative. The user of the account can thus loan money from the bank by letting the account balance become negative. When objects of class `AccountWithCredit` are created, a credit limit must be specified. Method `withdraw_money()` is rewritten for class `AccountWithCredit` so that it allows the account balance to be negative up to the specified credit limit.
- Objects of class `RestrictedAccount` do not allow excessive withdrawals from bank accounts. A withdrawal limit is specified when a `RestrictedAccount` object is created, and the rewritten method `withdraw_money()` of class `RestrictedAccount` guards that the withdrawal limit is not exceeded.

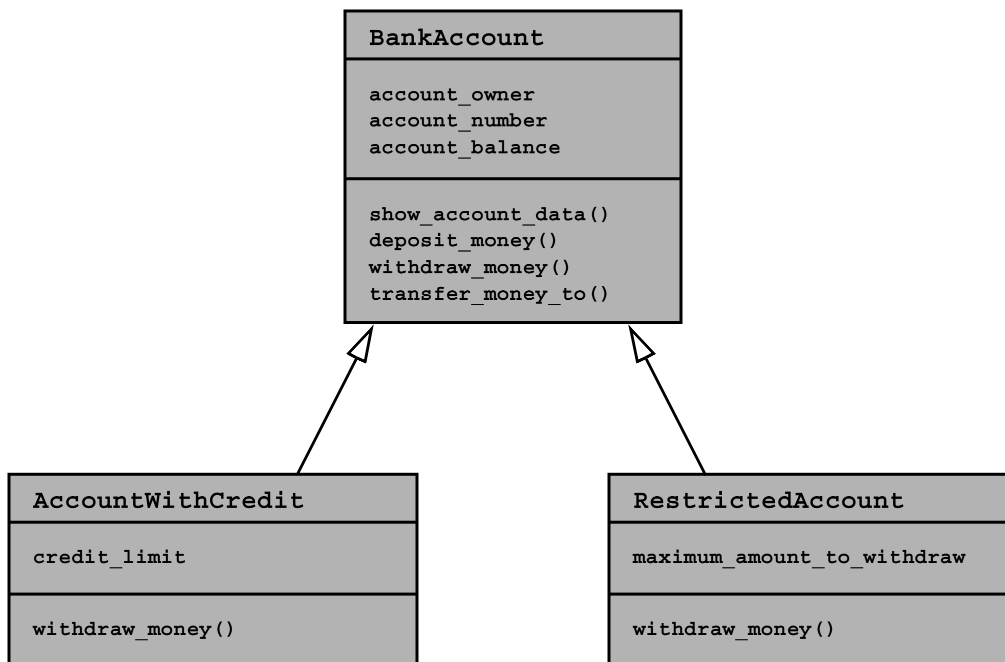

Figure 12-4. The class hierarchy in BankPolymorphic.js.

Some words about JavaScript constructors

If you do not provide a constructor for a class, a default constructor is generated automatically. The default constructor for a base class is

```
constructor()
{
}
```

The default constructor for a derived class is

```
constructor( ... given_parameters )
{
   super( ... given_parameters ) ;
}
```

The default constructor for a derived class thus passes its parameters to the constructor of of its superclass.

This means that you could derive a new class from class **`Animal`**, that was discussed earlier, in the following way

```
class DerivedAnimal extends Animal
{
}
```

and the objects of class **`DerivedAnimal`** could be created and used in the same way as **`Animal`** objects.