Class gnu.getopt.Getopt

```
java.lang.Object
   |
   +----gnu.getopt.Getopt
```

public class **Getopt**
extends Object

This is a Java port of GNU getopt, a class for parsing command line arguments passed to programs. It it based on the C getopt() functions in glibc 2.0.6 and should parse options in a 100% compatible manner. If it does not, that is a bug. The programmer's interface is also very compatible.

To use Getopt, create a Getopt object with a argv array passed to the main method, then call the getopt() method in a loop. It will return an int that contains the value of the option character parsed from the command line. When there are no more options to be parsed, it returns -1.

A command line option can be defined to take an argument. If an option has an argument, the value of that argument is stored in an instance variable called optarg, which can be accessed using the getOptarg() method. If an option that requires an argument is found, but there is no argument present, then an error message is printed. Normally getopt() returns a '?' in this situation, but that can be changed as described below.

If an invalid option is encountered, an error message is printed to the standard error and getopt() returns a '?'. The value of the invalid option encountered is stored in the instance variable optopt which can be retrieved using the getOptopt() method. To suppress the printing of error messages for this or any other error, set the value of the opterr instance variable to false using the setOpterr() method.

Between calls to getopt(), the instance variable optind is used to keep track of where the object is in the parsing process. After all options have been returned, optind is the index in argv of the first non-option argument. This variable can be accessed with the getOptind() method.

Note that this object expects command line options to be passed in the traditional Unix manner. That is, proceeded by a '-' character. Multiple options can follow the '-'. For example "-abc" is equivalent to "-a -b -c". If an option takes a required argument, the value of the argument can immediately follow the option character or be present in the next argv element. For example, "-cfoo" and "-c foo" both represent an option character of 'c' with an argument of "foo" assuming c takes a required argument. If an option takes an argument that is not required, then any argument must immediately follow the option character in the same argv element. For example, if c takes a non-required argument, then "-cfoo" represents option character 'c' with an argument of "foo" while "-c foo" represents the option character 'c' with no argument, and a first non-option argv element of "foo".

The user can stop getopt() from scanning any further into a command line by using the special argument "--" by itself. For example: "-a -- -d" would return an option character of 'a', then return -1 The "--" is discarded and "-d" is pointed to by optind as the first non-option argv element.

Here is a basic example of using Getopt:

```
 Getopt g = new Getopt("testprog", argv, "ab:c::d");
 //
 int c;
 String arg;
 while ((c = g.getopt()) != -1)
```

```
  {
    switch(c)
      {
         case 'a':
         case 'd':
           System.out.print("You picked " + (char)c + "\n");
           break;
           //
         case 'b':
         case 'c':
           arg = g.getOptarg();
           System.out.print("You picked " + (char)c +
                            " with an argument of " +
                            ((arg != null) ? arg : "null") + "\n");
           break;
           //
         case '?':
           break; // getopt() already printed an error
           //
         default:
           System.out.print("getopt() returned " + c + "\n");
      }
  }
```

In this example, a new Getopt object is created with three params. The first param is the program name. This is for printing error messages in the form "program: error message". In the C version, this value is taken from argv[0], but in Java the program name is not passed in that element, thus the need for this parameter. The second param is the argument list that was passed to the main() method. The third param is the list of valid options. Each character represents a valid option. If the character is followed by a single colon, then that option has a required argument. If the character is followed by two colons, then that option has an argument that is not required.

Note in this example that the value returned from getopt() is cast to a char prior to printing. This is required in order to make the value display correctly as a character instead of an integer.

If the first character in the option string is a colon, for example ":abc::d", then getopt() will return a ':' instead of a '?' when it encounters an option with a missing required argument. This allows the caller to distinguish between invalid options and valid options that are simply incomplete.

In the traditional Unix getopt(), -1 is returned when the first non-option charcter is encountered. In GNU getopt(), the default behavior is to allow options to appear anywhere on the command line. The getopt() method permutes the argument to make it appear to the caller that all options were at the beginning of the command line, and all non-options were at the end. For example, calling getopt() with command line args of "-a foo bar -d" returns options 'a' and 'd', then sets optind to point to "foo". The program would read the last two argv elements as "foo" and "bar", just as if the user had typed "-a -d foo bar".

The user can force getopt() to stop scanning the command line with the special argument "--" by itself. Any elements occuring before the "--" are scanned and permuted as normal. Any elements after the "--" are returned as is as non-option argv elements. For example, "foo -a -- bar -d" would return option 'a' then -1. optind would point to "foo", "bar" and "-d" as the non-option argv elements. The "--" is discarded by getopt().

There are two ways this default behavior can be modified. The first is to specify traditional Unix getopt() behavior (which is also POSIX behavior) in which scanning stops when the first non-option argument encountered. (Thus "-a foo bar -d" would return 'a' as an option and have "foo", "bar", and "-d" as non-option elements). The second is to allow options anywhere, but to return all elements in the order they occur on the command line. When a non-option element is ecountered, an integer 1 is returned and the value of the non-option element is stored in optarg is if it were the argument to that option. For example, "-a foo -d", returns first 'a', then 1 (with optarg set to "foo") then 'd' then -1. When this "return in order" functionality is enabled, the only way to stop getopt() from scanning all command line elements is to use the special "--" string by itself as described above. An example is "-a foo -b -- bar", which would return 'a', then integer 1 with optarg set to "foo", then 'b', then -1. optind would then point to "bar" as the first non-option argv element. The "--" is discarded.

The POSIX/traditional behavior is enabled by either setting the property "gnu.posixly_correct" or by putting a '+' sign as the first character of the option string. The difference between the two methods is that setting the gnu.posixly_correct property also forces certain error messages to be displayed in POSIX format. To enable the "return in order" functionality, put a '-' as the first character of the option string. Note that after determining the proper behavior, Getopt strips this leading '+' or '-', meaning that a ':' placed as the second character after one of those two will still cause getopt() to return a ':' instead of a '?' if a required option argument is missing.

In addition to traditional single character options, GNU Getopt also supports long options. These are preceeded by a "--" sequence and can be as long as desired. Long options provide a more user-friendly way of entering command line options. For example, in addition to a "-h" for help, a program could support also "--help".

Like short options, long options can also take a required or non-required argument. Required arguments can either be specified by placing an equals sign after the option name, then the argument, or by putting the argument in the next argv element. For example: "--outputdir=foo" and "--outputdir foo" both represent an option of "outputdir" with an argument of "foo", assuming that outputdir takes a required argument. If a long option takes a non-required argument, then the equals sign form must be used to specify the argument. In this case, "--outputdir=foo" would represent option outputdir with an argument of "foo" while "--outputdir foo" would represent the option outputdir with no argument and a first non-option argv element of "foo".

Long options can also be specified using a special POSIX argument format (one that I highly discourage). This form of entry is enabled by placing a "W;" (yes, 'W' then a semi-colon) in the valid option string. This causes getopt to treat the name following the "-W" as the name of the long option. For example, "-W outputdir=foo" would be equivalent to "--outputdir=foo". The name can immediately follow the "-W" like so: "-Woutputdir=foo". Option arguments are handled identically to normal long options. If a string follows the "-W" that does not represent a valid long option, then getopt() returns 'W' and the caller must decide what to do. Otherwise getopt() returns a long option value as described below.

While long options offer convenience, they can also be tedious to type in full. So it is permissible to abbreviate the option name to as few characters as required to uniquely identify it. If the name can represent multiple long options, then an error message is printed and getopt() returns a '?'.

If an invalid option is specified or a required option argument is missing, getopt() prints an error and returns a '?' or ':' exactly as for short options. Note that when an invalid long option is encountered, the optopt variable is set to integer 0 and so cannot be used to identify the incorrect option the user entered.

Long options are defined by LongOpt objects. These objects are created with a contructor that takes four params: a String representing the object name, a integer specifying what arguments the option takes (the value is one of LongOpt.NO_ARGUMENT, LongOpt.REQUIRED_ARGUMENT, or LongOpt.OPTIONAL_ARGUMENT), a StringBuffer flag object (described below), and an integer value (described below).

To enable long option parsing, create an array of LongOpt's representing the legal options and pass it to the Getopt() constructor. WARNING: If all elements of the array are not populated with LongOpt objects, the getopt() method will throw a NullPointerException.

When getopt() is called and a long option is encountered, one of two things can be returned. If the flag field in the LongOpt object representing the long option is non-null, then the integer value field is stored there and an integer 0 is returned to the caller. The val field can then be retrieved from the flag field. Note that since the flag field is a StringBuffer, the appropriate String to integer converions must be performed in order to get the actual int value stored there. If the flag field in the LongOpt object is null, then the value field of the LongOpt is returned. This can be the character of a short option. This allows an app to have both a long and short option sequence (say, "-h" and "--help") that do the exact same thing.

With long options, there is an alternative method of determining which option was selected. The method getLongind() will return the the index in the long option array (NOT argv) of the long option found. So if multiple long options are configured to return the same value, the application can use getLongind() to distinguish between them.

Here is an expanded Getopt example using long options and various techniques described above:

```
int c;
String arg;
LongOpt[] longopts = new LongOpt[3];
//
StringBuffer sb = new StringBuffer();
longopts[0] = new LongOpt("help", LongOpt.NO_ARGUMENT, null, 'h');
longopts[1] = new LongOpt("outputdir", LongOpt.REQUIRED_ARGUMENT, sb, 'o');
longopts[2] = new LongOpt("maximum", LongOpt.OPTIONAL_ARGUMENT, null, 2);
//
Getopt g = new Getopt("testprog", argv, "-:bc::d:hW;", longopts);
g.setOpterr(false); // We'll do our own error handling
//
while ((c = g.getopt()) != -1)
  switch (c)
    {
       case 0:
         arg = g.getOptarg();
         System.out.println("Got long option with value '" +
                            (char)(new Integer(sb.toString())).intValue()
                            + "' with argument " +
                            ((arg != null) ? arg : "null"));
         break;
         //
       case 1:
         System.out.println("I see you have return in order set and that " +
                            "a non-option argv element was just found " +
                            "with the value '" + g.getOptarg() + "'");
         break;
         //
       case 2:
         arg = g.getOptarg();
         System.out.println("I know this, but pretend I didn't");
         System.out.println("We picked option " +
                            longopts[g.getLongind()].getName() +
                          " with value " +
                          ((arg != null) ? arg : "null"));
         break;
         //
       case 'b':
         System.out.println("You picked plain old option " + (char)c);
         break;
         //
       case 'c':
       case 'd':
         arg = g.getOptarg();
         System.out.println("You picked option '" + (char)c +
                            "' with argument " +
                            ((arg != null) ? arg : "null"));
         break;
         //
       case 'h':
         System.out.println("I see you asked for help");
         break;
         //
       case 'W':
         System.out.println("Hmmm. You tried a -W with an incorrect long " +
                            "option name");
         break;
         //
       case ':':
         System.out.println("Doh! You need an argument for option " +
                            (char)g.getOptopt());
         break;
         //
       case '?':
         System.out.println("The option '" + (char)g.getOptopt() +
                          "' is not valid");
         break;
         //
       default:
         System.out.println("getopt() returned " + c);
         break;
    }
//
for (int i = g.getOptind(); i < argv.length ; i++)
  System.out.println("Non option argv element: " + argv[i] + "\n");
```

There is an alternative form of the constructor used for long options above. This takes a trailing boolean flag. If set to false, Getopt performs identically to the example, but if the boolean flag is true then long options are allowed to start with a single '-' instead of "--". If the first character of the option is a valid short option character, then the option is treated as if it were the short option. Otherwise it behaves as if the option is a long option. Note that the name given to this option - long_only - is very counter-intuitive. It does not cause only long options to be parsed but instead enables the behavior described above.

Note that the functionality and variable names used are driven from the C lib version as this object is a port of the C code, not a new implementation. This should aid in porting existing C/C++ code, as well as helping programmers familiar with the glibc version to adapt to the Java version even if it seems very non-Java at times.

In this release I made all instance variables protected due to overwhelming public demand. Any code which relied on optarg, opterr, optind, or optopt being public will need to be modified to use the appropriate access methods.

Please send all bug reports, requests, and comments to arenn@urbanophile.com.

Version:
    1.0.3
Author:
    Roland McGrath (roland@gnu.ai.mit.edu), Ulrich Drepper (drepper@cygnus.com), Aaron M. Renn (arenn@urbanophile.com)
See Also:
    LongOpt

Constructor Index

- **Getopt**(String, String[], String)
  Construct a basic Getopt instance with the given input data.
- **Getopt**(String, String[], String, LongOpt[])
  Construct a Getopt instance with given input data that is capable of parsing long options as well as short.
- **Getopt**(String, String[], String, LongOpt[], boolean)
  Construct a Getopt instance with given input data that is capable of parsing long options and short options.

Method Index

- **getLongind**()
  Returns the index into the array of long options (NOT argv) representing the long option that was found.
- **getopt**()
  This method returns a char that is the current option that has been parsed from the command line.
- **getOptarg**()
  For communication from `getopt' to the caller.
- **getOptind**()
  optind it the index in ARGV of the next element to be scanned.
- **getOptopt**()
  When getopt() encounters an invalid option, it stores the value of that option in optopt which can be retrieved with this method.
- **setArgv**(String[])
  Since in GNU getopt() the argument vector is passed back in to the function every time, the caller can swap out argv on the fly.
- **setOpterr**(boolean)
  Normally Getopt will print a message to the standard error when an invalid option is encountered.
- **setOptind**(int)
  This method allows the optind index to be set manually.
- **setOptstring**(String)
  In GNU getopt, it is possible to change the string containg valid options on the fly because it is passed as an argument to getopt() each time.

Constructors

Getopt

```
public Getopt(String progname,
              String argv[],
              String optstring)
```

Construct a basic Getopt instance with the given input data. Note that this handles "short" options only.

Parameters:
progname - The name to display as the program name when printing errors
argv - The String array passed as the command line to the program.
optstring - A String containing a description of the valid args for this program

Getopt

```
public Getopt(String progname,
              String argv[],
              String optstring,
              LongOpt long_options[])
```

Construct a Getopt instance with given input data that is capable of parsing long options as well as short.

Parameters:
progname - The name to display as the program name when printing errors
argv - The String array passed as the command ilne to the program
optstring - A String containing a description of the valid short args for this program

long_options - An array of LongOpt objects that describes the valid long args for this program

Getopt

```
public Getopt(String progname,
              String argv[],
              String optstring,
              LongOpt long_options[],
              boolean long_only)
```

Construct a Getopt instance with given input data that is capable of parsing long options and short options. Contrary to what you might think, the flag 'long_only' does not determine whether or not we scan for only long arguments. Instead, a value of true here allows long arguments to start with a '-' instead of '--' unless there is a conflict with a short option name.

Parameters:

progname - The name to display as the program name when printing errors
argv - The String array passed as the command ilne to the program
optstring - A String containing a description of the valid short args for this program
long_options - An array of LongOpt objects that describes the valid long args for this program
long_only - true if long options that do not conflict with short options can start with a '-' as well as '--'

Methods

setOptstring

```
public void setOptstring(String optstring)
```

In GNU getopt, it is possible to change the string containg valid options on the fly because it is passed as an argument to getopt() each time. In this version we do not pass the string on every call. In order to allow dynamic option string changing, this method is provided.

Parameters:

optstring - The new option string to use

getOptind

```
public int getOptind()
```

optind it the index in ARGV of the next element to be scanned. This is used for communication to and from the caller and for communication between successive calls to `getopt'. When `getopt' returns -1, this is the index of the first of the non-option elements that the caller should itself scan. Otherwise, `optind' communicates from one call to the next how much of ARGV has been scanned so far.

setOptind

```
public void setOptind(int optind)
```

This method allows the optind index to be set manually. Normally this is not necessary (and incorrect usage of this method can lead to serious lossage), but optind is a public symbol in GNU getopt, so this method was added to allow it to be modified by the caller if desired.

Parameters:

optind - The new value of optind

● setArgv

```
public void setArgv(String argv[])
```

Since in GNU getopt() the argument vector is passed back in to the function every time, the caller can swap out argv on the fly. Since passing argv is not required in the Java version, this method allows the user to override argv. Note that incorrect use of this method can lead to serious lossage.

Parameters:
argv - New argument list

● getOptarg

```
public String getOptarg()
```

For communication from `getopt' to the caller. When `getopt' finds an option that takes an argument, the argument value is returned here. Also, when `ordering' is RETURN_IN_ORDER, each non-option ARGV-element is returned here. No set method is provided because setting this variable has no effect.

● setOpterr

```
public void setOpterr(boolean opterr)
```

Normally Getopt will print a message to the standard error when an invalid option is encountered. This can be suppressed (or re-enabled) by calling this method. There is no get method for this variable because if you can't remember the state you set this to, why should I?

● getOptopt

```
public int getOptopt()
```

When getopt() encounters an invalid option, it stores the value of that option in optopt which can be retrieved with this method. There is no corresponding set method because setting this variable has no effect.

● getLongind

```
public int getLongind()
```

Returns the index into the array of long options (NOT argv) representing the long option that was found.

● getopt

```
public int getopt()
```

This method returns a char that is the current option that has been parsed from the command line. If the option takes an argument, then the internal variable 'optarg' is set which is a String representing the the value of the argument. This value can be retrieved by the caller using the getOptarg() method. If an invalid option is found, an error message is printed and a '?' is returned. The name of the invalid option character can be retrieved by calling the getOptopt() method. When there are no more options to be scanned, this method returns -1. The index of first non-option element in argv can be retrieved with the getOptind() method.

Returns:
Various things as described above
