# Jukebox class

The Jukebox class can be used to play simple melodies on the NXT brick. Create a new Java class named ‘Jukebox’ in your Lejos-enabled project. Replace the code in that class by the code below and compile. Then use in your project as described in the comments. Add new melodies (and share with us).

**import** lejos.nxt.Sound;

/\* @version Nov. 2008

 \* @author Bert Wachsmuth

 \*

 \* <h1>Jukebox</h1>

 \*

 \* This class defines notes as frequencies as well as various songs (defined as

 \* 2-D arrays) and let's you play those songs in their own thread. A song can

 \* be looped, stopped, and switched any time. Note that the thread continues to

 \* run once the class is instantiated - you need to explicitly stop the thread by

 \* calling 'poweroff'. Once the thread is turned off, however, you can no longer

 \* play any songs.

 \*

 \* Usage:

 \*

 \* (1) Create a new Jukebox (which starts the thread)

 \*

 \* Jukebox player = new Jukebox();

 \*

 \* (2) To play a song, use the 'play' method

 \*

 \* player.play(Jukebox.STARWARS\_INTRO, false);

 \*

 \* where the first parameter is the melody and the second one is a boolean

 \* value indicating whether to loop the melody (true) or not (false). If a

 \* song was already playing it will be turned off and the new one plays.

 \*

 \* (3) To stop the song, use the 'off' method

 \*

 \* player.off();

 \*

 \* (4) When you no longer need the music player, call 'poweroff'

 \*

 \* player.poweroff()

 \*

 \* It will terminate the thread and you can no longer use it.

 \*

 \* Please note that the Lejos Java Virtual Machine (JVM) does not do any

 \* garbage collection. You should therefore be careful making 'new' objects.

 \* Thus, it would create memory problems if each new song would use a new

 \* thread, so this class starts only one thread to play songs as needed,

 \* one song at a time.

 \*/

**public** **class** Jukebox **extends** Thread

{

 // Frequencies for defined notes. Add more as needed.

 **final** **static** **int** *C7* = 2093;

 **final** **static** **int** *B6* = 1975;

 **final** **static** **int** *AIS6* = 1865;

 **final** **static** **int** *A6* = 1760;

 **final** **static** **int** *GIS6* = 1661;

 **final** **static** **int** *G6* = 1568;

 **final** **static** **int** *FIS6* = 1480;

 **final** **static** **int** *F6* = 1397;

 **final** **static** **int** *E6* = 1318;

 **final** **static** **int** *DIS6* = 1244;

 **final** **static** **int** *D6* = 1175;

 **final** **static** **int** *CIS6* = 1109;

 **final** **static** **int** *C6* = 1046;

 **final** **static** **int** *B5* = 988;

 **final** **static** **int** *AIS5* = 932;

 **final** **static** **int** *A5* = 880;

 **final** **static** **int** *GIS5* = 831;

 **final** **static** **int** *G5* = 784;

 **final** **static** **int** *FIS5* = 740;

 **final** **static** **int** *F5* = 698;

 **final** **static** **int** *E5* = 659;

 **final** **static** **int** *DIS5* = 622;

 **final** **static** **int** *D5* = 587;

 **final** **static** **int** *CIS5* = 554;

 **final** **static** **int** *C5* = 523;

 // Length of a quarter note

 **final** **static** **int** *BEAT* = 250;

 // Sleep time in ms before resuming thread when nothing is playing

 **final** **static** **int** *SLEEP* = 100;

 // Indicates a pause

 **final** **static** **int** *PAUSE* = -1;

 // Sets the default volume

 **final** **static** **int** *VOLUME* = Sound.*VOL\_MAX*;

 // A sample melody

 **final** **static** **int** *STARWARS\_INTRO*[][] =

 {

 {*C5*, 2\**BEAT*}, {*F5*, 4\**BEAT*}, {*C6*, 2\**BEAT*},

 {*AIS5*, *BEAT*}, {*A5*, *BEAT*}, {*G5*, *BEAT*},

 {*F6*, 4\**BEAT*}, {*C6*, 2\**BEAT*}, {*AIS5*, *BEAT*},

 {*A5*, *BEAT*}, {*G5*, *BEAT*}, {*F6*, 4\**BEAT*},

 {*C6*, 2\**BEAT*}, {*AIS5*, *BEAT*}, {*A5*, *BEAT*},

 {*AIS5*,*BEAT*}, {*G5*, 6\**BEAT*}, {*PAUSE*, 4\**BEAT*}

 };

 // More melodies

 **final** **static** **int** *JAWS*[][] =

 {

 {*D5*, *BEAT*}, {*DIS5*, *BEAT*}, {*D5*, *BEAT*}, {*E5*, *BEAT*}

 };

 // Simple sound effect

 **final** **static** **int** *BACKUP*[][] =

 {

 {*C6*, *BEAT*}, {*PAUSE*,*BEAT*}, {*C6*, *BEAT*}, {*PAUSE*, *BEAT*}

 };

 **private** **int**[][] melody = **null**;

 **private** **boolean** playing = **true**;

 **private** **boolean** looping = **true**;

 /\*

 \* Constructor sets up variable and starts the thread

 \*/

 **public** Jukebox()

 {

 **super**();

 melody = **null**;

 playing = **true**;

 start();

 }

 /\*

 \* Plays the melody, either in a loop or just once.

 \*/

 **public** **void** play(**int**[][] melody, **boolean** looping)

 {

 **try**

 {

 off();

 *sleep*(*BEAT*);

 }

 **catch**(InterruptedException ex)

 {

 }

 **this**.melody = melody;

 **this**.looping = looping;

 }

 /\*

 \* Stops the currently playing song, if any

 \*/

 **public** **void** off()

 {

 melody = **null**;

 }

 /\*

 \* Shuts down the thread

 \*/

 **public** **void** poweroff()

 {

 melody = **null**;

 playing = **false**;

 }

 /\*

 \* Called automatically when thread starts. Do NOT cll this

 \* method directly

 \*/

 **public** **void** run()

 {

 **while** (playing)

 {

 **try**

 {

 **if** (melody != **null**)

 playTheMelody();

 **else**

 *sleep*(*SLEEP*);

 }

 **catch**(InterruptedException iex)

 {

 }

 }

 }

 /\*

 \* Private method to handle the actual playing of the melody.

 \* Since this method is private, you can not call it directly -

 \* call 'play' instead.

 \*/

 **private** **void** playTheMelody() **throws** InterruptedException

 {

 **int** i = 0;

 **int** length = -1;

 **int** note = *A5*;

 **int** duration = *BEAT*;

 **if** (melody != **null**)

 length = melody.length;

 **while** (i < length)

 {

 **if** (melody == **null**)

 length = -1;

 **else**

 {

 **if** (melody != **null**)

 note = melody[i][0];

 **if** (melody != **null**)

 duration = melody[i][1];

 **if** (note != *PAUSE*)

 Sound.*playTone*(note, duration);

 *yield*();

 *sleep*(duration);

 i++;

 }

 }

 **if** (!looping)

 melody = **null**;

 }

}