

## Tutorial 7: Repeating TimerTasks

Contributed by D1st0rt  
 Saturday, 25 February 2006  
 Last Updated Tuesday, 30 May 2006

This is the second of two tutorials to demonstrate the use of TimerTasks. In this tutorial, we will go through how to use TimerTasks that repeat every so often. For this tutorial, we will be creating a new TWBot module, countdown, so make the file twbotcountdown.java in your bots/twbot folder. When a bot op uses !countdown, the module will send an arena message every second with the number of remaining seconds until time runs out. We also want to be able to !stop the currently running countdown.

```

package twcore.bots.twbot;

import java.util.TimerTask;
import twcore.core.*;

/**
 * Basic module that displays a countdown through arena messages.
 * @author D1st0rt
 * @version 06.02.25
 */
public class twbotcountdown extends TWBotExtension
{
  /** The module's countdown task */
  private Countdown countdown;
  /** Module Help message */
  private final String[] helpMsg = {
    [countdown]",
    "!countdown - starts a new countdown",
    "!stop - stops a currently running countdown"
  };

  /**
   * Creates a new instance of the countdown module
   */
  public twbotcountdown()
  {
  }

  /**
   * Event: Message
   * Called when a player sends a chat message
   * Handle commands
   */
  public void handleEvent(Message event)
  {
    if(event.getMessageType() == Message.PRIVATE_MESSAGE)
    {
      String name = m_botAction.getPlayerName(event.getPlayerID());
      String message = event.getMessage();
      if(m_opList.isER(name) && message.startsWith("!"))
        delegateCmd(name, message);
    }
  }

  /**
   * This takes the validated message, determines the command,
   * and passes it to the proper method
   * @param name the name of the player that send the command
   * @param message the text that the player sent
   */
  public void delegateCmd(String name, String message)
  {

```

```
String cmd = message.substring(1).toLowerCase();

if(cmd.startsWith("countdown "))
    c_Countdown(name, message.substring(11));
else if(cmd.equals("stop"))
    c_Stop(name, message);
}
```

This stuff is all the same, except that just for convenience we pass the substring starting at the 11th character of the message as the command parameters to the countdown method. By making sure that our string starts with "countdown " instead of just "countdown", we avoid the nasty `StringIndexOutOfBoundsException` we would get if the user didn't enter a number in the command.

```

    /**
    * Command: !countdown
    * Starts a new countdown with the specified number of seconds
    */
public void c_Countdown(String name, String message)
{
    if(m_opList.isER(name))
    {
        try{
            int seconds = Integer.parseInt(message);

            if(countdown != null)
                countdown.cancel();

            countdown = new Countdown(seconds);
            m_botAction.scheduleTaskAtFixedRate(countdown, 1000, 1000);

        }catch(Exception e)
        {
            m_botAction.sendPrivateMessage(name, "use !countdown ");
        }
    }
}

    /**
    * Command: !stop
    * Stops the currently running countdown
    */
public void c_Stop(String name, String message)
{
    if(m_opList.isER(name))
    {
        if(countdown != null)
        {
            countdown.cancel();
            m_botAction.sendPrivateMessage(name, "Countdown stopped");
        }
    }
}
}
```

When the user tells the bot !countdown, we have to figure out what number they entered. If they don't enter a number, we have to catch the exception and tell them to enter a number next time. We could also optionally defer the exception from no parameters here and just check for "countdown" in the previous code block. If they do in fact enter a number, we create a new Countdown with this number and start it going. Stopping the countdown is pretty self-explanatory, just make sure you check for a null countdown first.

```

        /**
    * Gets the help message(s) to be used by the TWBot !help command
    * @return the help message(s) for this module
    */
public String[] getHelpMessages()
{
    return helpMsg;
}

/**
* Stops any running tasks so the module can be unloaded.
*/
public void cancel()
{
    if(countdown != null)
        countdown.cancel();
}

```

We don't want the countdown to continue counting down after we unload the module, so make sure you cancel it first if it's running.

```

        /**
    * This class represents a task to remove a player from the list of
    * players recently entering the arena after 10 minutes
    */
private class Countdown extends TimerTask
{
    /** The remaining time to count down */
    private int secondsLeft;

    /**
    * Creates a new instance of the Countdown task
    * @param seconds the number of seconds to count down
    */
    public Countdown(int seconds)
    {
        secondsLeft = seconds;
    }

    /**
    * Execute this task: Count down to 0

```

```
*/
public void run()
{
    if(secondsLeft > 0)
    {
        m_botAction.sendArenaMessage("" + secondsLeft);
    }
    else if(secondsLeft == 0)
        m_botAction.sendArenaMessage("Time's up!");
    else
        this.cancel();

        secondsLeft--;
}
}
```

This is where all the real work gets done in this module. Every second when the countdown is executed, it sends an arena message of how much time is left. If there is no time left, it says "Time's up!". The remaining number of seconds is decremented each time and any value less than 0 will terminate the countdown so range checking for negative numbers is optional.