LA BIBLIOTHÈQUE SwitchPack
TUTORIEL

TABLE DES MATIÈRES

Que retrouve-t-on dans la bibliothèque SwitchPack?....2
À propos des interrupteurs....4
- Anatomie d'un interrupteur mécanique....4
- Connecter un interrupteur à l'Arduino....4

Utiliser la bibliothèque SwitchPack....5
La classe Debounce....5
- Les sources du problème....5
- L'approche....6
- L'algorithme....6
- L'utilisation de la classe....6

La classe Contact....7
La classe Click....8
La classe DoubleClick....8
La classe Toggle....9
La classe TimedClick....10
La classe Repeater....11
La classe ModeSwitch....11
La classe Encoder....12

Que retrouve-t-on dans la bibliothèque SwitchPack?

Cette bibliothèque contient 9 classes d'objets de type interrupteur :

1. **Debounce :** C'est un algorithme extrêmement rapide pour éliminer les rebonds que l'on retrouve lorsqu'on ouvre ou on ferme un interrupteur. Il élimine également les lectures qui pourraient être faussées par des interférences électromagnétiques. Toutes les autres classes de la bibliothèque héritent de cette classe.
   Pour en savoir plus, vous pouvez consulter l'article d'où est tiré l'algorithme :
   http://www.ganssle.com/debouncing.htm
   Vous pouvez également lire le tutoriel :
   https://github.com/j-bellavance/EdgeDebounceLite/blob/master/tutorials/tutoriel.pdf

2. **Contact :** Hérite de Debounce. Permet de créer un interrupteur qui, indépendamment qu'il soit monté en mode d'excursion haute (PULLUP) ou en mode d'excursion basse (PULLDOWN), peut signaler qu'il :
   est ouvert : .open()
   est fermé : .closed()
   vient de passer de fermé à ouvert : .fell()
   vient de passer d'ouvert à fermé : .rose()

3. **Click** : Hérite de Contact. De plus, l'interrupteur peut signaler qu'il :
   a été fermé et qu'il vient juste d'être ouvert (clic) : .clicked()

4. **DoubleClick :** Hérite de Click. De plus, l'interrupteur peut :
   signaler le nombre de « clics » effectués sur une période déterminée : .clickCount()
   varier la période d'admissibilité à un double-clic : .setLimit()
   être transformé en triple-clic, quadruple-clic… : .setMaxClics()

5. **Toggle :** Hérite de Click. De plus, il permet de transformer un interrupteur momentané en interrupteur ouvert/fermé. Il peut :
   vérifier et signaler son état actuel (ouvert ou fermé) : .readStatus()
   signaler son état actuel (ouvert ou fermé) : .getStatus()
   accepter un état initial : .setStatus()

6. **TimedClick :** Hérite de Click. De plus, il possède trois chronomètres. En conjonction avec la commande millis() de l'IDE de l'Arduino, Il peut signaler :
   le moment où il a été fermé la dernière fois : .wasLastRead()
   la durée durant laquelle il a été fermé : .clickTime()
   la durée écoulée depuis la dernière fois où il a été ouvert : .timeSinceLastClick()

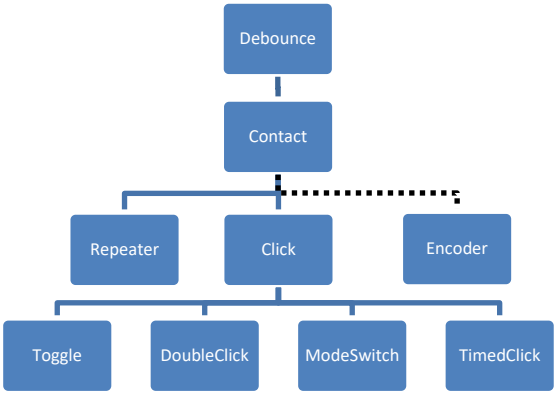
7. **Repeater :** Hérite de Contact. De plus, il se comporte comme les touches du clavier d'un ordinateur. Au moment où l'interrupteur est fermé, une première touche est envoyée à l'écran. Si l'interrupteur reste fermé suffisamment longtemps, la touche est envoyée par la suite à intervalle régulier jusqu'à ce que l'interrupteur soit ouvert. Cette classe retourne vrai lorsqu'il est temps d'envoyer la touche. Il est possible d'accomplir n'importe quelle action à ce moment (pas seulement envoyer une touche!) :
   Pour savoir si c'est le moment d'envoyer une touche, on demande : .repeatRequired()

Le délai avant l'envoi des autres touches peut être modifié : .setStart()
Le délai entre chaque envoi subséquent peut aussi être modifié : .setBurst()

8. **ModeSwitch :** Hérite de Click. De plus, il possède un compteur qui est incrémenté à chaque clic. Quand le compteur atteint un nombre prédéterminé, il est remis à zéro. Pour un interrupteur à 4 modes, la valeur retournée sera successivement : 0, 1, 2, 3, 0, 1, 2, 3, 0… Il peut :
   vérifier et signaler le mode : .readMode()
   signaler le mode : .getMode()
   remettre à zéro le compteur après un délai prédéterminé : .resetAfter()

9. **Encoder :** Cette classe n'hérite d'aucune classe. Elle utilise deux objets de la classe Contact. Elle représente un encodeur rotatif qui peut :
   signaler dans quel direction il a été tourné. Dans le sens antihoraire (CCW :-1), dans le sens horaire (CW :1) ou statique (0) : .getRotation()
   retourner la valeur du compteur interne : .getCount()
   accepter une nouvelle valeur au compteur : .setCount()
   remettre le compteur à zéro : .resetCounter()

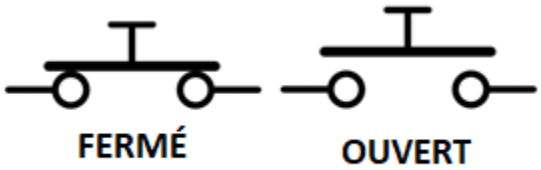
Le diagramme suivant montre les relations d'héritage entre les 9 classes de la bibliothèque. La classe **Encoder** est différente par le fait qu'elle n'hérite d'aucune autre classe, mais qu'elle possède deux objets de classe **Contact**.



À propos des interrupteurs

Anatomie d'un interrupteur mécanique

Un interrupteur mécanique (par opposition à l'interrupteur optique) est composé de deux ou plusieurs pièces conductrices. Une des pièces peut être positionnée pour être en contact avec l'autre. On dit alors que l'interrupteur est **fermé** (et conduit le courant).S les pièces ne sont pas en contact, on dit que l'interrupteur est **ouvert** (et le courant ne passe pas).



Connecter un interrupteur à l'Arduino

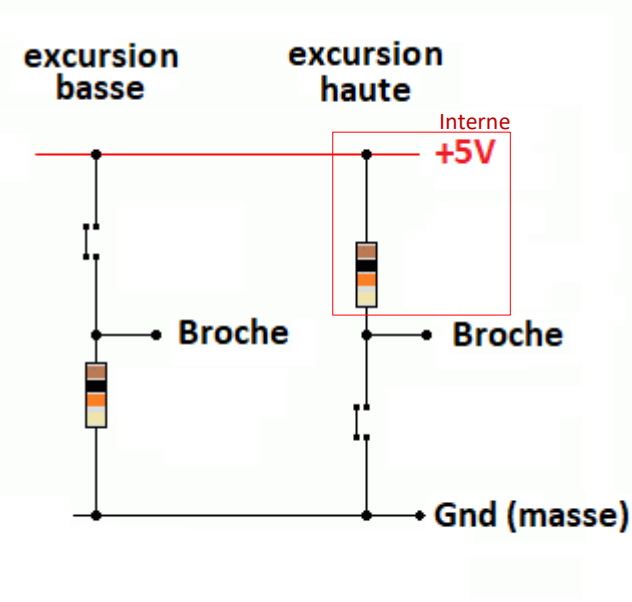
Un interrupteur est connecté à une broche d'Arduino. Celle-ci est capable de rapporter so état grâce à la commande :

```
digitalRead(4);
```

Si le voltage de la broche est entre 0 et 0,8 V, digitalRead() retourne LOW.
Si le voltage de la broche est entre 2,2 et 5 V, digitalRead() retourne HIGH.
Si le voltage de la broche est entre 0,9 et 2,1 V, digitalRead() retourne HIGH ou LOW, au hasard.

Il existe deux manières de connecter un interrupteur à un Arduino :

- En mode d'excursion haute (PULLUP)
- En mode d'excursion basse (PULLDOWN)



À gauche, l'interrupteur est connecté en mode d'excursion basse (PULLDOWN). Quand il est ouvert, la résistance nous assure que la broche sera à la masse (LOW). Quand il est fermé, +5V amène la broche à (HIGH). Une résistance d'environ 10KΩ est adéquate pour cet usage. La broche est initialisée dans la section setup() du sketch avec :

```
pinMode(4, INPUT)
```

À droite, l'interrupteur est connecté en mode d'excursion haute (PULLUP). Quand il est ouvert, +5V est relié à la broche, ce qui l'amène à (HIGH). Quand il est fermé, +5V est dirigé vers la masse, laissant la broche à (LOW). Nous utilisons la résistance interne de l'Arduino. La broche est initialisée dans la section setup() du sketch avec :

```
pinMode(4, INPUT_PULLUP)
```

Utiliser la bibliothèque SwitchPack

Pour utiliser n'importe quelle classe de SwitchPack, il suffit de l'inclure dans le sketch :

```
#include <SwitchPack.h>
```

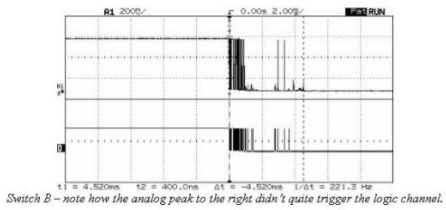
La classe Debounce

Les sources du problème

Un interrupteur est conçu pour interrompre le courant ou lui laisser le passage

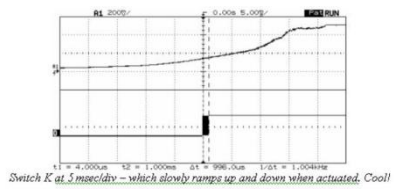
Malheureusement, il y a deux moments où l'interrupteur laisse passer le courant d'une manière aléatoire. C'est quand on l'ouvre ou quand on le ferme. Quand les contacts sont tout près un de l'autre, mais que l'interrupteur n'est pas tout à fait ouvert ou fermé. Les électrons peuvent migrer d'un contact vers l'autre se manière sporadique. On pourrait le comparer à des éclairs dans un orage. Plus les contacts sont près l'un de l'autre, plus il y aura d'éclairs laissant passer le courant de l'un à l'autre.

Voici un exemple, à l'oscilloscope, d'un interrupteur que l'on est en train de fermer[1] :



Switch B – note how the analog peak to the right didn't quite trigger the logic channel.

Dans le haut de l'image, on voit le voltage tel que mesuré. Dans le bas on voit le voltage retourné par un convertisseur analogue-digital. On peut voir les « éclairs », très fréquents au début, puis qui s'espacent jusqu'à ce que le convertisseur se stabilise à 0 V. Il se passe 2 millisecondes entre le moment où l'interrupteur commence à être relâché et celui où il est enfin stable. Si on met en boucle un digitalRead() pendant cette période, il retournera environ 650 lectures et on ne peut être certain d'aucune d'elles.



Switch K at 5 msec/div – which slowly ramps up and down when actuated. Cool!

Voici un autre exemple :
Cet interrupteur ne se comporte pas comme l'autre. Le voltage augmente progressivement. Si on regarde l'image du bas, il se passe quelque chose d'étrange. Pendant un certain temps le convertisseur analogique-digital est stable à 0 V. Puis, il devient très instable, passant de 5 V à 0 V très rapidement. Il se stabilise enfin à 5V, quand l'interrupteur est effectivement fermé. Cela tient aux spécifications du convertisseur analogique-digital. S'il mesure entre 0 et 0,8 V sur la broche, il retourne LOW. S'il mesure entre 2,2 et 5 V, il retourne HIGH. Entre 0,8 et 2,2 V, il retourne au hasard HIGH ou LOW, ne nous laissant pas savoir que c'est ce qu'il fait.

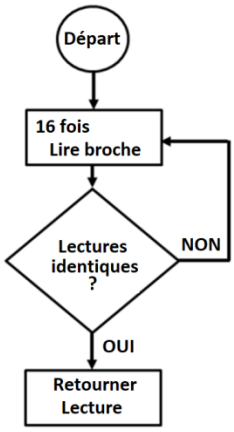
Les sources d'interférences électromagnétiques sont aussi la cause de bien des soucis. Choisir des câbles blindés est souvent une chose sensée à faire. Toutefois, avec des sources moyennes, Debounce est capable de les éliminer.

[1] Ces exemples sont tirés de : http://www.ganssle.com/debouncing.htm

L'approche

Comme nous venons de le voir, le comportement erratique ne se produit que lorsque l'interrupteur est en voie de se fermer ou de s'ouvrir. L'approche en sera une de patience. Au lieu de ne prendre qu'une lecture de la broche, nous en prendrons plusieurs successivement. Supposons que l'on décide de prendre 16 lectures successives aussi rapprochées que possible (le tout se passe en environ 90 millionièmes de secondes). On vérifie ensuite que toutes les lectures sont identiques (toutes HIGH ou toutes LOW). Si c'est le cas, les probabilités que le résultat soit valide est grand. Si, par contre, il y a des lectures HIGH parmi des LOW, on a la certitude que l'on est en phase de transition. Comme on l'a dit précédemment, il suffit d'être patient et d'attendre que la tempête cesse. On va reprendre des séries de 16 lectures jusqu'à ce qu'elles concordent toutes (16 HIGH ou 16 LOW).

L'algorithme[2]



1. Lire (entre 1 et 32) fois la broche
2. Si toutes les lectures sont identiques, retourner la lecture
3. Si non, reprendre à l'étape 1

Par défaut, les broches sont lues en rafales de 16. Il est toutefois possible de changer la taille de ces rafales entre 1 (aucune stabilisation du signal) et 32 (nombre maximal de lectures faites en rafale ≈180 µs).

Pour la plupart des interrupteurs, 16 lectures par rafale sont suffisantes pour stabiliser le signal. Selon le type d'interrupteur ou de l'application qu'on désire en faire, il est possible d'en changer la valeur. Par exemple, j'ai plusieurs encodeurs et j'obtiens de meilleurs résultats avec ceux-ci avec des rafales de 8 lectures.

L'utilisation de la classe

La classe Debounce permet de stabiliser le signal sur n'importe quelle broche digitale de l'Arduino. On créée un objet Debounce ainsi :

```
Debounce debounce;
```

Une fois l'objet créé, On peut lire la broche ainsi :

```
byte bouton = debounce.pin(4);
```

On doit se rappeler que :

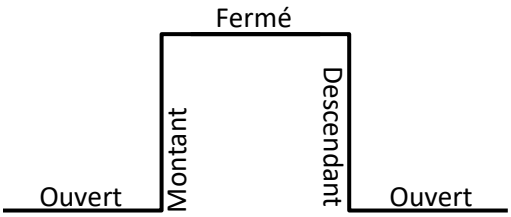
- En mode d'excursion basse, si le bouton est fermé, la broche retourne HIGH.
- En mode d'excursion haute, si le bouton est fermé, la broche retourne LOW.

Le répertoire « examples » montre comment gérer les deux types de connections possibles d'un interrupteur (excursion haute et basse). DebounceAnyPin.ino

[2] Cet algorithme peut être retrouvé ici : http://www.ganssle.com/debouncing-pt2.htm

La classe Contact

La classe Contact hérite de la classe Debounce.
Lors de l'utilisation d'un interrupteur, celui-ci passe par quatre phases :



Ouvert : le voltage est à 0
Montant : le voltage passe de 0 à +5V
Fermé : le voltage est à +5V
Descendant : le voltage passe de +5V à 0V

La classe contact permet d'utiliser une broche de l'Arduino pour lire l'interrupteur sans considération qu'il a été monté en mode d'excursion haute ou basse. Les résultats seront identiques dans les deux cas. Il suffit de mentionner le mode lors de la création de l'objet :

```
Contact broche(4, PULLDOWN)
Contact broche(4, PULLUP)
Contact broche(4)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Il y a quatre méthodes qui permettent de vérifier dans quelle phase est l'interrupteur :

```
if (broche.closed()) {…;}    //Phase fermé
if (broche.open()) {…;}      //Phase Ouvert
if (broche.rose()) {…;}      //Phase Montant
if (broche.fell()) {…;}      //Phase Descendant
```

Une broche peut indépendamment lire .closed() ou .open(). Par contre, les méthodes .rose() et .fell() exigent toute l'attention de la broche. Il n'y a qu'une seule lecture parmi toutes celles utilisées pour détecter cette phase qui retourne vrai. Toutes les autres retourneront faux.

Si la broche doit être capable de signaler plus d'une phase, La structure doit se construire ainsi :

```
broche.update();
if (broche.getClosed()) {…;}    //Phase fermé
if (broche.getOpen()) {…;}      //Phase Ouvert
if (broche.getRose()) {…;}      //Phase Montant
if (broche.getFell()) {…;}      //Phase Descendant
```

Le répertoire « examples » contient trois sketches qui permettent de prendre en main la classe Contact :

1. Les deux types de connections possibles d'un interrupteur (excursion haute et basse) sont utilisés afin de montrer que les méthodes de Contact retournent la même valeur, peu importe le type de connexion dans (Contact.ino).
2. La méthode .fell() est utilisée pour démontrer la vitesse d'exécution de l'Arduino dans (ContactSpeed.ino)

3. Les quatre phases qui composent l'appui et le relâchement de l'interrupteur sont illustrées grâce à 3 DEL dans (SwitchCycles.ino).

La classe Click

La classe Click hérite de la classe Contact. Un objet Click peut faire tout ce que fait un objet Contact.

L'exemple Contact.ino a montré à quel point Arduino est rapide. Beaucoup rapide que nos doigts. Afin de s'assurer qu'une partie d'un sketch ne s'exécute qu'une fois sur un appui sur l'interrupteur, la classe Click ne surveille que la phase descendante. Ainsi, on est assuré que l'interrupteur a bel et bien été fermé, et que le sketch est averti aussitôt que l'interrupteur est ré-ouvert.

Pour créer un objet Click :

```
Click broche(4, PULLDOWN)
Click broche(4, PULLUP)
Click broche(4)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Pour savoir si l'objet a été actionné on procède ainsi :

```
If (broche.clicked()) {…;}
```

Le répertoire « examples » contient deux sketches qui permettent de prendre en main la classe Click :

1. Un clic fait clignoter la DEL 13 de l'Arduino dans (Click.ino).
2. Un tableau de cinq interrupteurs est utilisé pour montrer comment gérer plusieurs interrupteurs dans un même sketch dans (ArrayOfClicks).

La classe DoubleClick

La classe DoubleClick hérite de la classe Click. Un objet DoubleClick peut faire tout ce que fait un objet Click, et tout ce que fait Contact.

Un double-clic est décrit comme appuyer et relâcher un interrupteur deux fois à l'intérieur d'un intervalle fixe de temps. L'objet DoubleClick retourne combien de fois l'interrupteur a été appuyé et relâché.

Pour créer un objet DoubleClick avec un intervalle de 500 millisecondes(½ s) :

```
Click broche(4, PULLDOWN, 500)
Click broche(4, PULLUP, 500)
Click broche(4, 500)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Pour prendre action selon le nombre de clics effectués on procède ainsi :

```
Switch (broche.clickCount()) {
  case 1: {…; break;}
  case 2: {…; break;}
}
```

Pour accepter des triple-clics, quadruples-clics, dans la section setup() du sketch, après l'initialisation de la broche :

```
broche.begin();
broche.setMaxClicks(3);
```

Le répertoire « examples » contient deux sketches qui permettent de prendre en main la classe DoubleClick :

1. La DEL 13 de l'Arduino clignote le nombre de fois que l'interrupteur a été cliqué dans (DoubleClick.ino).
2. Le même principe est appliqué dans (TripleClick.ino).

La classe Toggle

La classe Toggle hérite de la classe Click. Un objet Toggle peut faire tout ce que fait un objet Click, et tout ce que fait Contact.

Cette classe nous permet de transformer un interrupteur momentané en interrupteur ouvert/fermé (ON/OFF). Il possède un indicateur qui est mis à jour lors des lectures de la broche. Cet indicateur peut prendre les valeurs ON ou OFF.

Pour créer un objet Toggle :

```
Toggle broche(4, PULLDOWN)
Toggle broche(4, PULLUP)
Toggle broche(4,)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Pour agir selon l'état de la broche on procède ainsi :

```
if (broche.readStatus() == ON) {…;}
if (broche.readStatus() == OFF) {…;}
```

Pour obtenir l'état de l'objet Toggle sans lire la broche on procède ainsi :

```
byte etatToggle = broche.getStatus();
```

L'état initial de l'objet Toggle peut être spécifié ainsi :

```
broche.setStatus(ON);
broche.setStatus(OFF);
```

Le répertoire « examples » contient un sketch qui permet de prendre en main la classe Toggle en changeant l'état de la DEL 13 selon les clicks d'un interrupteur (Toggle.ino).

La classe TimedClick

La classe TimedClick hérite de la classe Click. Un objet TimedClick peut faire tout ce que fait un objet Click, et tout ce que fait Contact.

Cette classe ajoute trois chronomètres (à comparer avec millis()) à la classe Click :

- Le premier indique le moment où l'objet a été consulté la dernière fois.
- Le second sert à signaler combien de temps l'interrupteur est resté dans la phase fermée.
- Le troisième sert à signaler depuis combien de temps l'interrupteur est passé à la phase descendante.

Pour créer un objet TimedClick :

```
TimedClick broche(4, PULLDOWN)
TimedClick broche(4, PULLUP)
TimedClick broche(4,)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

L'utilisation d'un objet TimedClick se fait de la même manière qu'un objet Click :

```
if (broche.Clicked()) {…;}
```

Par la suite, si on désire savoir quand l'interrupteur a été consulté la dernière fois :

```
unsigned long lue = broche.wasLastRead();
```

Si on désire savoir combien de temps l'interrupteur est resté dans la phase fermée :

```
unsigned long duree = broche.clickTime();
```

Si on désire savoir depuis combien de temps l'interrupteur est passé à la phase descendante :

```
unsigned long relachee = broche.timeSinceLastClick();
```

Le répertoire « examples » contient un sketch qui permet de prendre en main la classe TimedClick en affichant sur le Moniteur Série les différents délais (TimedClick.ino).

La classe Repeater

La classe Repeater hérite de la classe Contact. Un objet Repeater peut faire tout ce que fait un objet Contact.

La classe Repeater se comporte comme les touches du clavier d'un ordinateur. Au moment où l'interrupteur est fermé, une première touche est envoyée à l'écran. Si l'interrupteur reste fermé suffisamment longtemps, la touche est envoyée par la suite à intervalle régulier jusqu'à ce que l'interrupteur soit ouvert. Cette classe retourne vrai lorsqu'il est temps d'envoyer la touche. Il est possible d'accomplir n'importe quelle action à ce moment, comme contrôler un moteur, simuler un accélérateur,… (pas seulement envoyer une touche!) :

Pour créer un objet Repeater, on doit signaler l'intervalle à observer avant que la touche soit répétée (ici 500 millisecondes) ainsi que le délai entre l'envoi des touches subséquentes(ici 50 millisecondes) :

```
Repeater broche(4, PULLDOWN, 500, 50)
Repeater broche(4, PULLUP, 500, 50)
Repeater broche(4,500, 50)  //Si le mode n'est pas mentionné, c'est le mode PULLUP
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Pour savoir si le moment d'envoyer une touche est atteint, on procède ainsi :

```
if (broche.repeatRequired()) {…;}
```

Il est aussi possible de changer les deux délais :

```
broche.setStart(500)
broche.setBurst(50);
```

Le répertoire « examples » contient un sketch qui permet de prendre en main la classe Repeater en affichant sur le Moniteur Série l'envoi de la touche 'A' (TimedClick.ino).

La classe ModeSwitch

La classe ModeSwitch hérite de la classe Click. Un objet ModeSwitch peut faire tout ce que fait un objet Click, et tout ce que fait Contact.

La classe ModeSwitch possède un compteur qui est incrémenté à chaque clic. Quand le compteur atteint un nombre prédéterminé, il est remis à zéro. Pour un interrupteur à 4 modes, la valeur retournée sera successivement : 0, 1, 2, 3, 0, 1, 2, 3, 0…

Pour créer un objet ModeSwitch avec six modes :

```
ModeSwitch broche(4, PULLDOWN, 6)
ModeSwitch broche(4, PULLUP, 6)
ModeSwitch broche(4, 6)  //Si le mode n'est pas mentionné, c'est le mode PULLUP qui est
activé.
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
broche.begin();
```

Pour prendre action selon le mode de l'objet:

```
Switch (broche.readMode()) {
  case 0: {…; break;}
  case 1: {…; break;}
  case 2: {…; break;}
  case 3: {…; break;}
}
```

Pour lire le mode courant:

```
byte leMode = broche.getMode();
```

L'objet ModeSwitch possède un chronomètre qui lui permet de revenir automatiquement au mode 0 après un intervalle donné. Le chronomètre est remis à zéro à chaque fois que l'interrupteur change de phase. La remise à 0 du mode ne s'effectue que si l'interrupteur n'a pas changé de mode dans l'intervalle qui lui est imparti. Pour activer cette option, dans la section setup() du sketch, après l'initialisation de la broche on indique l'intervalle en millisecondes :

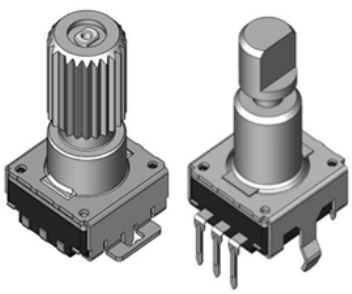
```
broche.begin();
broche.reserAfter(1000);
```

Le répertoire « examples » contient deux sketches qui permettent de prendre en main la classe ModeSwitch :

1. Un clic fait clignoter la DEL 13 de l'Arduino selon le mode dans (Click.ino).
2. Ce sketch utilise :
   un tableau d'objets Repeater;
   un objet Contact;
   un objet ModeSwitch.
   pour simuler un mini clavier de trois touches, avec une touche (MAJUSCULE) et une touche (FIXE MAJUSCULE) dans (MiniKeyboard.ino).

La classe Encoder

C'est la dernière classe de la bibliothèque SwitchClass. Elle n'hérite d'aucune classe. Elle utilise plutôt deux objets contact pour lire un encodeur rotatif.



Les encodeurs rotatifs peuvent être tournés de manière horaire ou antihoraire indéfiniment. Il est habituellement pourvu de détentes qui donnent une rétroaction indiquant qu'un pas a été accompli dans un sens ou dans l'autre.

À l'intérieur de l'encodeur se trouvent deux interrupteurs qui permettent de déterminer dans quel sens l'encodeur a été tourné. On l'appelle aussi codeur quadratique.

L'encodeur est muni de trois broches. Pour le connecter à un Arduino, procéder comme suit :
Broche A de l'encodeur vers une **broche digitale** de l'Arduino;
Broche Com de l'encodeur vers la broche **Gnd** de l'Arduino (masse);
Broche B de l'encodeur vers une **broche digitale** de l'Arduino;

Pour créer un objet Encodeur il faut préciser quelles sont les broches A et B :

```
Encoder encodeur(7, 8)
```

Dans la section setup() du sketch, l'initialisation se fait ainsi :

```
encodeur.begin();
```

Si les broches de l'encodeur ont été relies en mode d'excursion basse (PULLDOWN), dans la section setup() du sketch, à la suite de l'initialisation de la broche :

```
encodeur.begin();
encodeur.setModes(PULLDOWN, PULLDOWN);
```

Si, à l'usage, on réalise que l'encodeur ne réagit qu'une fois sur deux, c'est que l'on est en présence d'un encodeur qi ne possède que 2 états par clic. La majorité des encodeurs possèdent 4 états par clic. Il est facile d'y remédier en ajoutant, dans la section setup() du sketch, à la suite de l'initialisation de la broche :

```
encodeur.begin();
encodeur.stepsPerClick(2);
```

Pour prendre action selon la direction dans laquelle a été tourné l'encodeur:

```
int rotation = encodeur.getRotation();
if (rotation == CW){…;}
if (rotation == CCW){…;}
```

L'objet Encoder possède un compteur qui est automatiquement incrémenté ou décrémenté selon la direction dans laquelle l'encodeur a été tourné. Pour obtenir sa valeur on procède ainsi :

```
int compteur = encodeur.getCount();
```

Le compteur peut être initialisé ou remis à zéro ainsi :

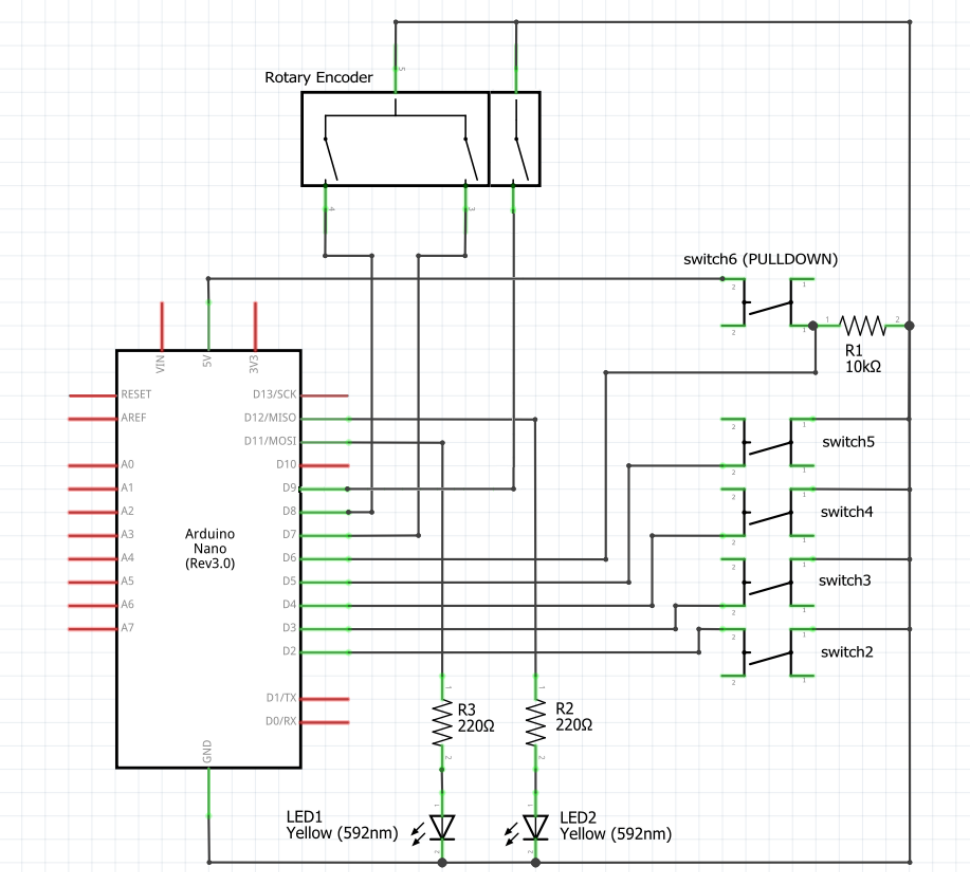
```
encodeur.setCount(50);
encodeur.resetCount();
```

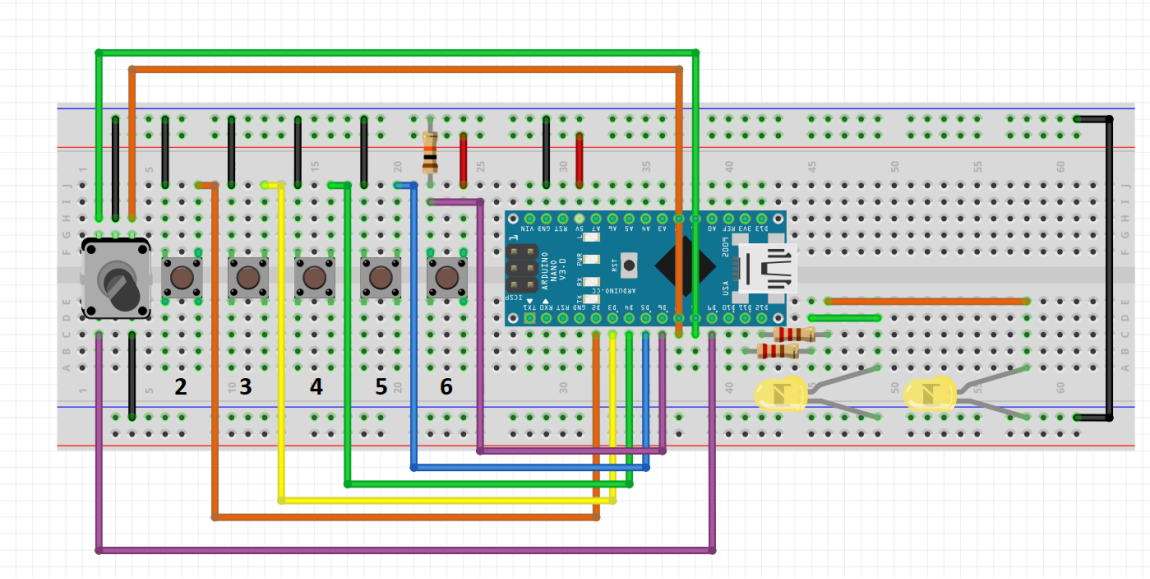
Le répertoire « examples » contient deux sketches qui permettent de prendre en main la classe Encoder :

1. Le premier affiche au Moniteur série 1 si la rotation est dans le sens horaire ou -1 si elle est dans le sens antihoraire (Encoder.ino).
2. Certains encodeurs viennent avec un interrupteur qui est activé en pressant l'encodeur. Voir (EncoderWithMode.ino).

J'espère sincèrement que la bibliothèque SwitchPack vous sera utile pour vos projets
Jacques Bellavance

Annexe A : Shéma et platine d'exploration pour tous les exemples :





Les interrupteurs 2 à 5 sont montés en mode d'excursion haute. L'interrupteur 6 est monté en mode d'excursion basse. La résistance de l'interrupteur 6 est de 10KΩ. Les deux résistances des DEL sont de 220Ω.