ENSEIRB-MATMECA

MISE EN ŒUVRE DE JAVA EMBARQUE JAVA ME SUR OBJET CONNECTE RPI

Patrice KADIONIK
kadionik.enseirb-matmeca.fr

TABLE DES MATIERES

1. BUT DES TRAVAUX PRATIQUES

Ces Travaux Pratiques ont pour but de présenter la mise en œuvre de Java embarqué sur un objet connecté construit autour d'une carte Raspberry Pi.

Il s'agit de voir la mise en œuvre pratique de Java embarqué dans l'environnement Java ME (*Java Micro Edition*) ou J2ME adapté aux systèmes embarqués et aux objets connectés de l'IoT (*Internet of Things*).

Des applications Java J2ME ou *MIDlets* (ou *IMlet*) seront développées pour illustrer certains aspects de ce langage et seront testés sur la carte cible Raspberry Pi (RPi) que l'on appellera carte rpi-java.

On abordera les points suivants :

- Application *MIDlet* « Hello World » pour la prise en main de l'IDE NetBeans.
- Boutons poussoirs de la carte rpi-java : scrutation, *listener*, interface `PinListener`, classe Java.
- Leds de la carte rpi-java.
- Afficheur LCD.
- Capteur de température DS1624.
- *Threads* Java. Interface `Runnable`.
- Sémaphore binaire. Synchronisation de *threads*. Mot clé `synchronized`, méthodes `wait()` et `notify()`.
- Test d'un serveur Web Java embarqué.
- Serveur Web Java embarqué pour récupérer à distance la température de la carte rpi-java.
- Miniprojet : mise en œuvre de toutes les entrées/sorties de la carte rpi-java.

Mots clés : Java embarqué, Java ME, J2ME, IDE, NetBeans, *MIDlet*, *IMlet*, Raspberry Pi

2. INFORMATIONS ESSENTIELLES SUR LA CARTE RPI-JAVA

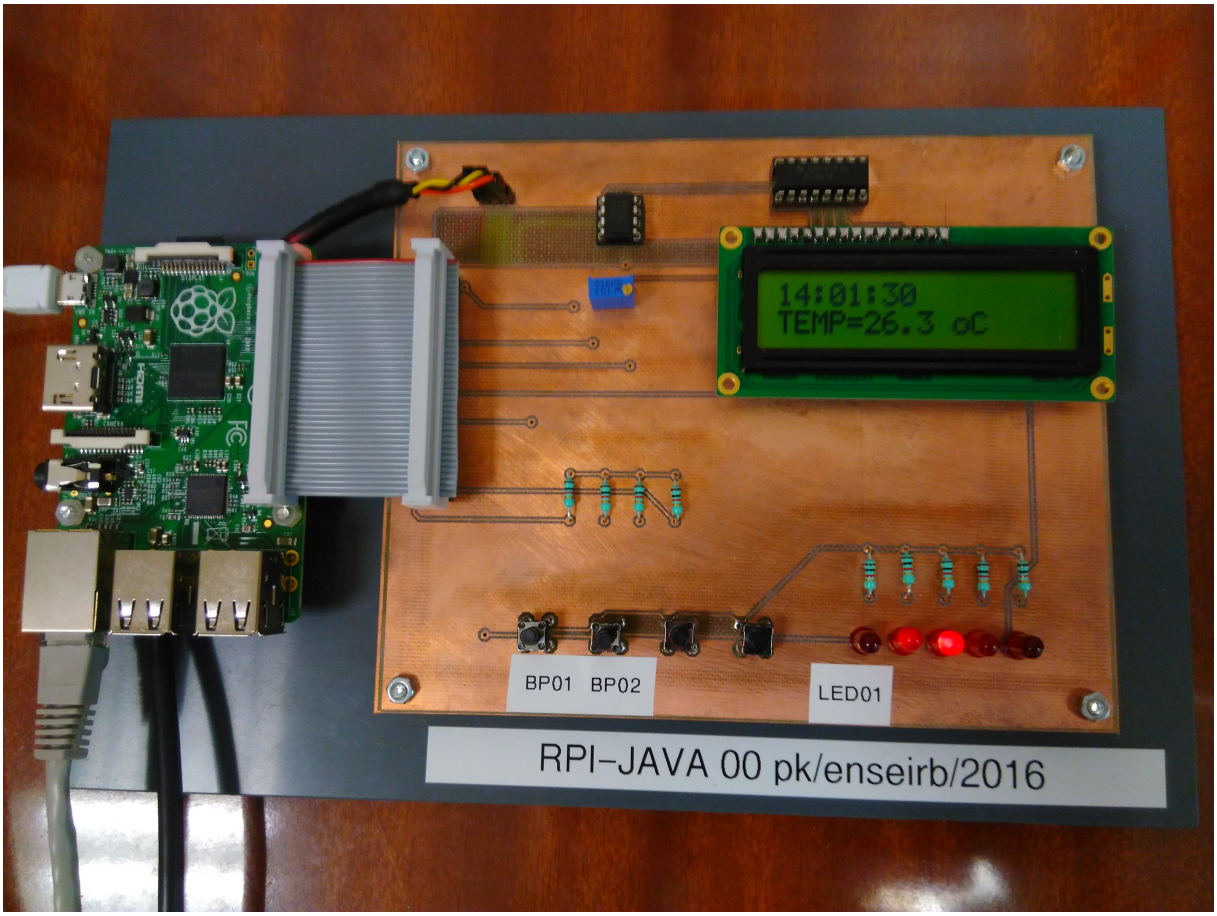
La carte cible rpi-java est une carte « maison » construite autour d'une carte Raspberry Pi 3B. Elle est complétée d'une carte d'entrées/sorties (E/S) connectée à l'aide d'un connecteur 2x20 broches au connecteur d'E/S de la carte RPi.

La carte d'E/S de le carte cible rpi-java possède :

- 5 leds : LED1 à LED5.
- 4 boutons poussoirs : BP1 à BP4.
- Un afficheur LCD 2x16 caractères interfacé sur le bus I2C de la carte RPi.
- Un capteur de température numérique DS1624 interfacé sur le bus I2C de la carte RPi.

Le schéma électronique de la carte d'E/S de la carte rpi-java est donné en annexe 1.

La photo est de la carte rpi-java est donnée ci-après.



Carte cible rpi-java

On notera sur le tableau suivant la correspondance entre le numéro de la broche du connecteur 2x20 de la carte RPi et le nom du signal de la carte d'E/S.

Numéro de broche connecteur RPi	Nom du signal carte d'E/S	Fonction
1	V33	3,3 V
2	VCC	5 V
3	SDA	Bus I2C
5	SCL	Bus I2C
6	GND noir	RS232
7	PIN_7	BP1
8	TxD jaune	RS232
10	RxD orange	RS232
11	PIN_11	BP2
12	PIN_12	LED2
13	PIN_13	BP4
15	PIN_15	BP3
16	PIN_16	LED3
18	PIN_18	LED4
22	PIN_22	LED5
25	GND	Masse
26	PIN_26	LED1

Connecteur 2x20 broches et signaux de la carte rpi-java

Les capteurs qui sont associés à ces signaux seront contrôlés par des applications Java ME *MIDlet*.

3. INFORMATIONS ESSENTIELLES SUR L'IDE NETBEANS

NetBeans est un atelier de développement logiciel libre ou IDE (*Integrated Design Entry*) qui permet entres autres de développer des applications Java et plus particulièrement des applications Java ME.

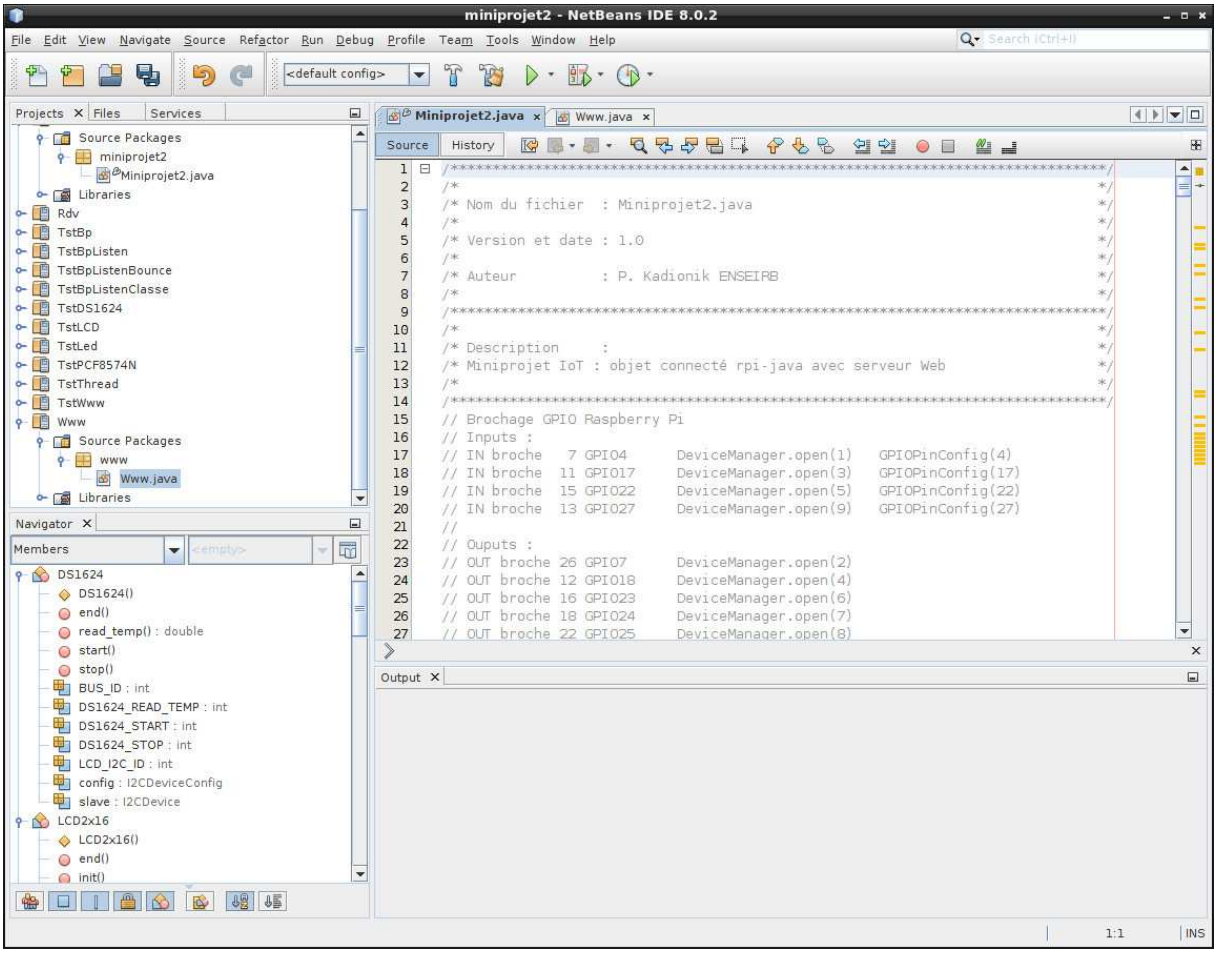
NetBeans sera utilisé sous Linux.

L'application Java ME sera créée en langage Java puis compilée et empaquetée pour créer une *MIDlet*. Un agent s'exécutant sur le PC de développement permet d'envoyer la *MIDlet* à un autre agent s'exécutant sur la carte cible rpi-java qui la fera alors exécuter par la machine virtuelle Java ME.

Il convient de lancer au préalable la JVM (*Java Virtual Machine*) et l'agent de la carte rpi-java (en ligne de commande par la console Linux de la carte cible rpi-java).

La configuration réseau du PC hôte et de la carte cible rpi-java étant préalablement faite, il n'y a pas besoin de s'en préoccuper.

La figure suivante présente l'interface graphique de NetBeans.



Interface graphique de NetBeans

On y retrouve 4 zones principales :

- La zone d'édition du fichier source.
- La zone des projets Java ME.
- La zone donnant des informations sur les objets Java.
- La zone de traces d'exécution *output*.

On notera le point essentiel suivant : la compilation du projet Java ME, la génération de la *MIDlet* et le téléchargement/exécution dans la carte cible rpi-java se fait en cliquant sur le bouton avec l'icône en forme de triangle vert…

Les traces d'exécution de la *MIDlet* apparaissent sur la console de la carte rpi-java mais aussi dans la zone de traces d'exécution *output*.

La maîtrise de NetBeans se fera alors au fur et à mesure des exercices…

On notera le point important suivant : il n'y a pas de logique entre le numéro d'une broche d'un signal d'E/S du connecteur 2x20 et le numéro utilisé pour son initialisation en langage Java dans le fichier source de la *MIDlet*.

Le tableau suivant donne néanmoins la correspondance, résultat d'expérimentations, car la documentation sur Java ME et son emploi sur la carte RPi est inexacte en grande partie.

Broche	Numéro de broche connecteur RPi	Fonction	Nom logique	Méthodes Java d'initialisation
IN	7	BP1	GPIO4	DeviceManager.open(1) GPIOPinConfig(4)
IN	11	BP2	GPIO17	DeviceManager.open(3) GPIOPinConfig(17)
IN	15	BP3	GPIO22	DeviceManager.open(5) GPIOPinConfig(22)
IN	13	BP4	GPIO27	DeviceManager.open(9) GPIOPinConfig(27)
OUT	26	LED1	GPIO7	DeviceManager.open(2)
OUT	12	LED2	GPIO18	DeviceManager.open(4)
OUT	16	LED3	GPIO23	DeviceManager.open(6)
OUT	18	LED4	GPIO24	DeviceManager.open(7)
OUT	22	LED5	GPIO25	DeviceManager.open(8)

Méthodes Java d'initialisation des E/S

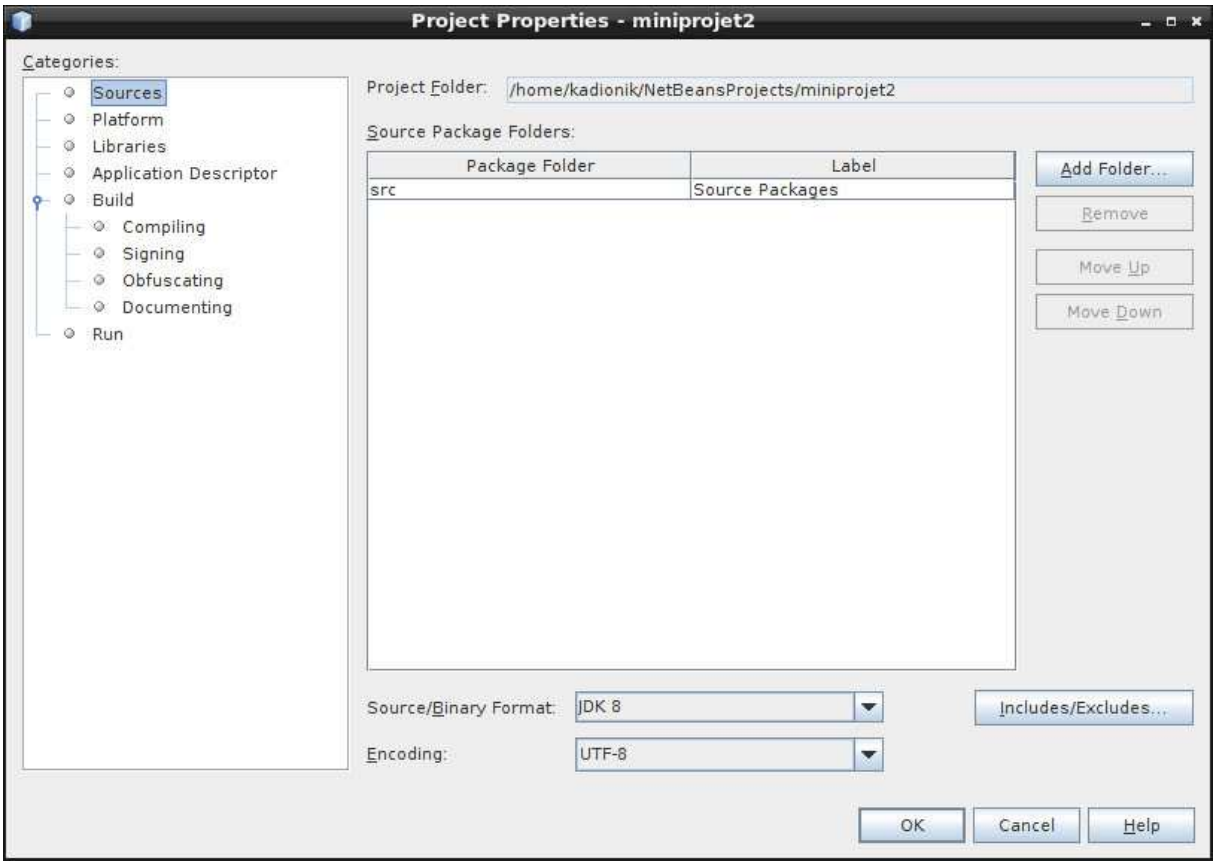
Concernant les capteurs sur le bus I2C, ils sont repérés par leur identificateur I2C suivant le tableau suivant :

I2C Id	capteur
0x20	Afficheur LCD 2x16 caractères
0x48	Capteur de température DS1624

Identificateur des capteurs I2C de la carte cible rpi-java

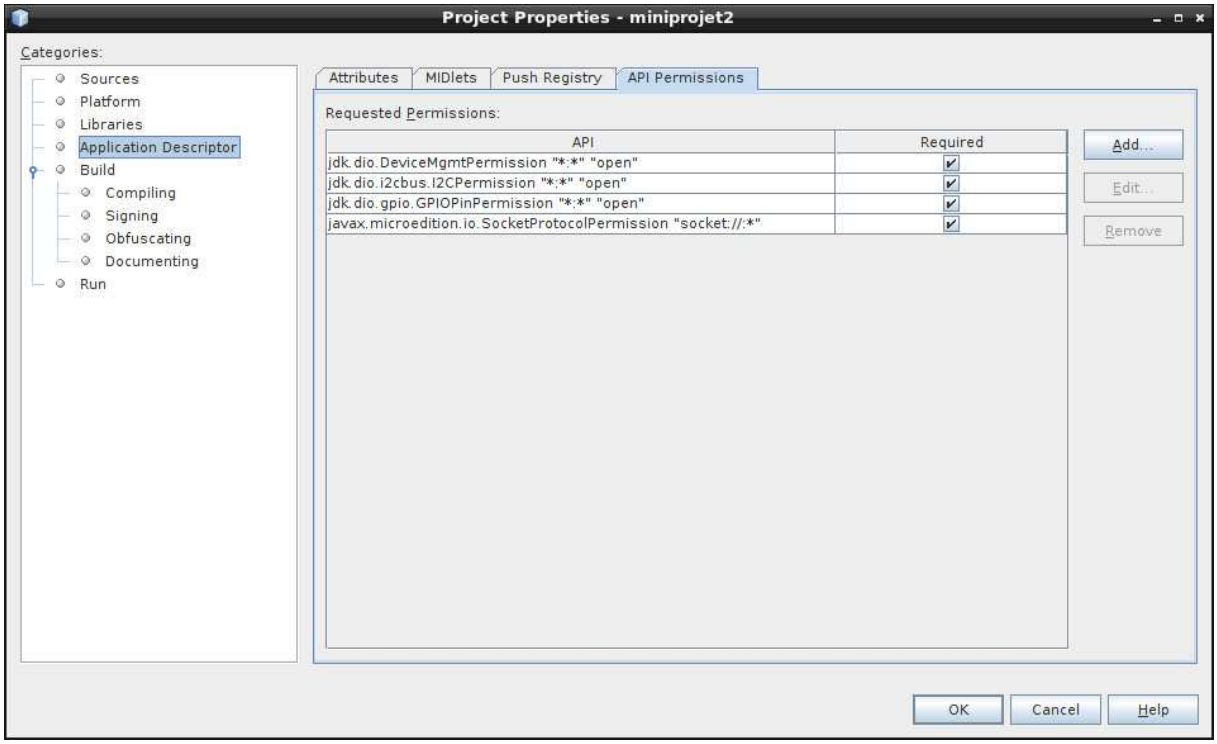
On notera enfin cet autre point important : une *MIDlet* doit avoir les droits autorisés pour accéder aux E/S qu'elle désire contrôler.

Par un clic droit sur un projet NetBeans, on accède au menu *Properties*.



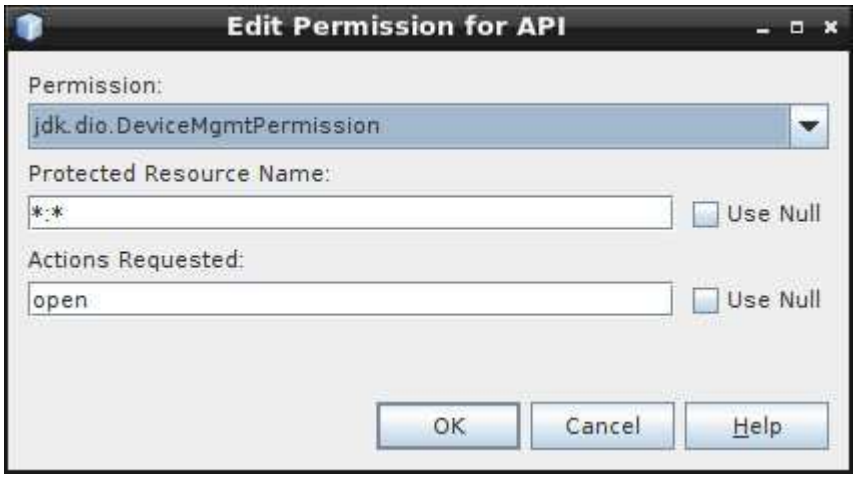
Permissions de la *MIDlet* pour accéder aux E/S (1)

On cliquera ensuite sur l'item *Application Descriptor* puis sur l'onglet *API Permissions*.



Permissions de la *MIDlet* pour accéder aux E/S (2)

On créera ensuite les permissions requises en fonction de quel type d'E/S on désire contrôler.



Permissions de la *MIDlet* pour accéder aux E/S (3)

Les permissions à valider sont présentées dans le tableau suivant :

E/S	Permission	Valeur 1	Valeur 2
BP et leds	`jdk.dio.DeviceMgmtPermission`	`*:*`	`open`
BP et leds	`jdk.dio.gpio.GPIOPinPermission`	`*:*`	`open`
Afficheur et capteur I2C	`jdk.dio.i2cbus.I2CPermission`	`*:*`	`open`
Serveur *socket*	`javax.microedition.io.SocketProtocolPermission`	`socket://:*`	

Liste des permissions utiles pour une *MIDlet*

Par la suite, on adoptera les conventions suivantes :
Commande Linux PC hôte :

```
host% commande Linux
```

Commande Linux carte cible :

```
rpi-java$ commande Linux
```

4. TP 0 : PRISE EN MAIN

- Démarrer le PC sous Linux. Se connecter sous le nom **se01**, mot de passe : **se01** ☺ pour le groupe 1 et sous le nom **se02**, mot de passe : **se02** ☺ pour le groupe 2.

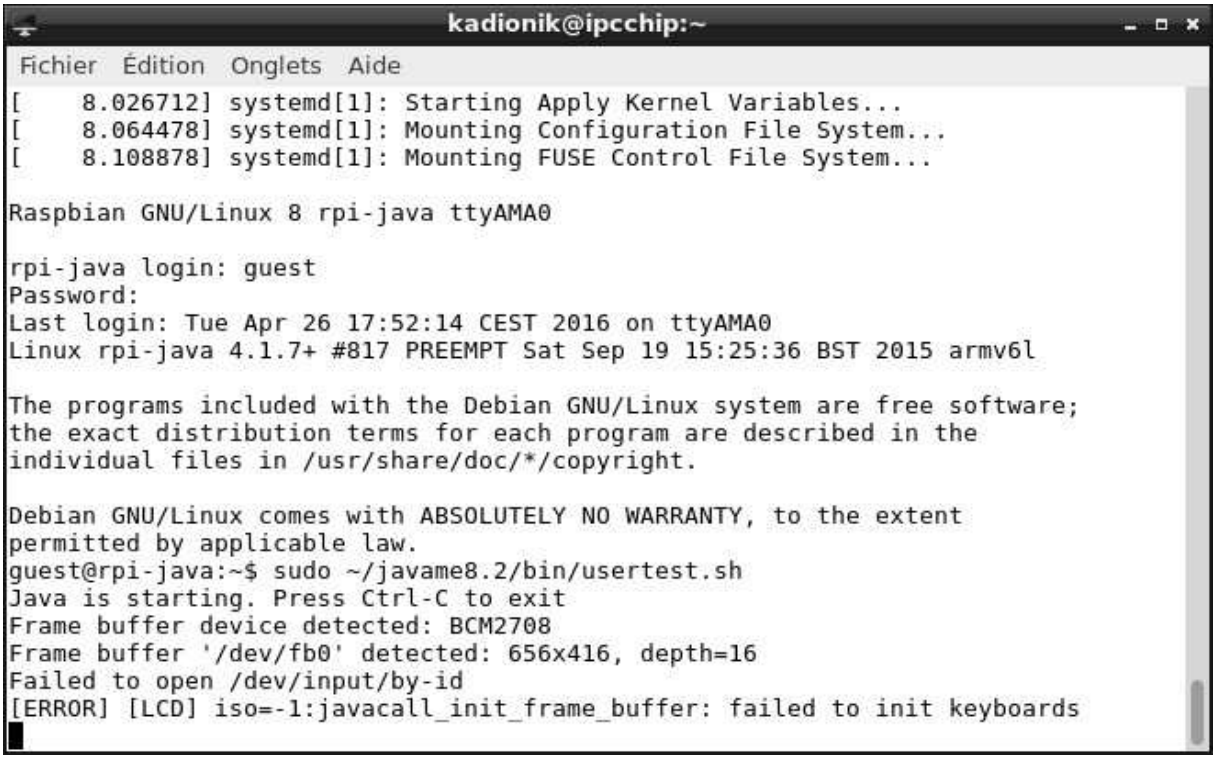
- Se placer dans son répertoire de travail :
  `host% cd`

- Etablir le schéma de l'environnement de développement :
  - Matériels.
  - Liaisons : série, réseau…
  - Logiciels et OS utilisés.
  - Adresses IP du PC de développement (hôte ou *host*) et de la carte cible (cible ou *target*).

- Se connecter à la carte cible rpi-java en utilisant l'outil `minicom` :
  `host% minicom`
  Pour sortir de `minicom`, il suffit de taper la combinaison de touches : CTRL A, Z pour accéder au menu et taper q pour quitter.
  Se connecter sur la carte cible rpi-java sous le nom **guest**, mot de passe : **guest** ☺.

- Récupérer les fichiers squelettes des projets Java ME des différents exercices du fichier *tar ball* `NetBeansProjects.tgz` :
  ```
  host% cd
  host% cp /home/kadionik/ NetBeansProjects.tgz ~
  host% tar -xvzf  NetBeansProjects.tgz
  ```

- Lancer la JVM de la carte cible rpi-java comme indiqué sur la figure suivante :
  `rpi-java$ sudo ~/javame8.2/bin/usertest.sh`



Lancement de la Java VM sur la carte cible rpi-java

5. EX 1 : PROJET HELLO

Le but de cet exercice est de créer l'application classique « Hello World ! ».

- Créer le projet de *MIDlet* `Hello` en suivant le tutorial pas à pas donné en images ci-après.
- Télécharger dans la carte cible rpi-java et tester.

Indications :

- On utilisera la méthode Java `System.out.println()` comme équivalent du `printf()` en langage C.

Les figures suivantes montrent la création pas à pas du projet `Hello`. La méthode s'appliquera ensuite pour tous les autres exercices…

On vérifiera que l'on est bien connecté avec l'agent de la carte cible rpi-java en dialoguant avec l'agent du PC. Il suffira de cliquer sur l'icône *Device Manager* en bas à droite du bureau.

Device Connection Manager (1)

On activera le lien avec la carte cible en cliquant sur l'icône en forme de plus et on précisera l'adresse IP de la carte cible rpi-java (voir annexe 3) et son nom logique, ici RPI.

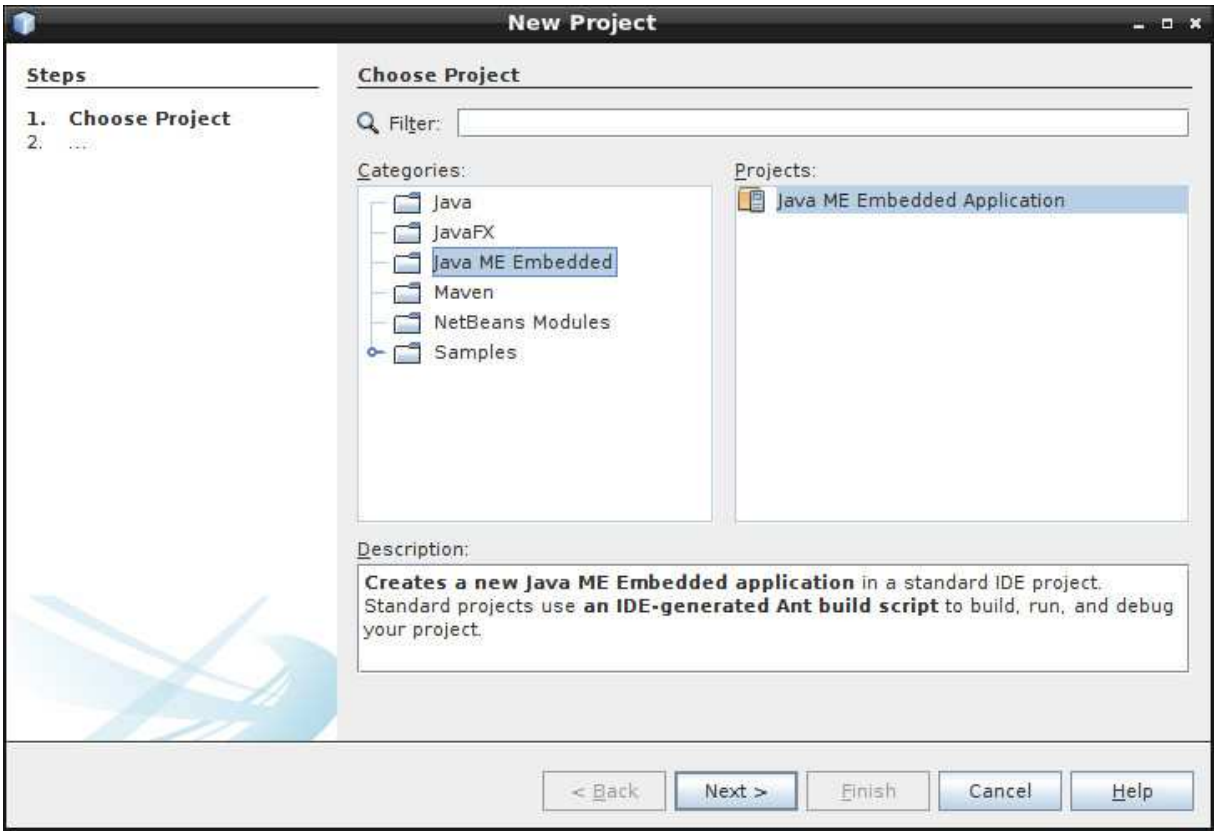
Device Connection Manager (2)

Si tout va bien, on doit être connecté avec la carte cible rpi-java :

Device Connection Manager (3)

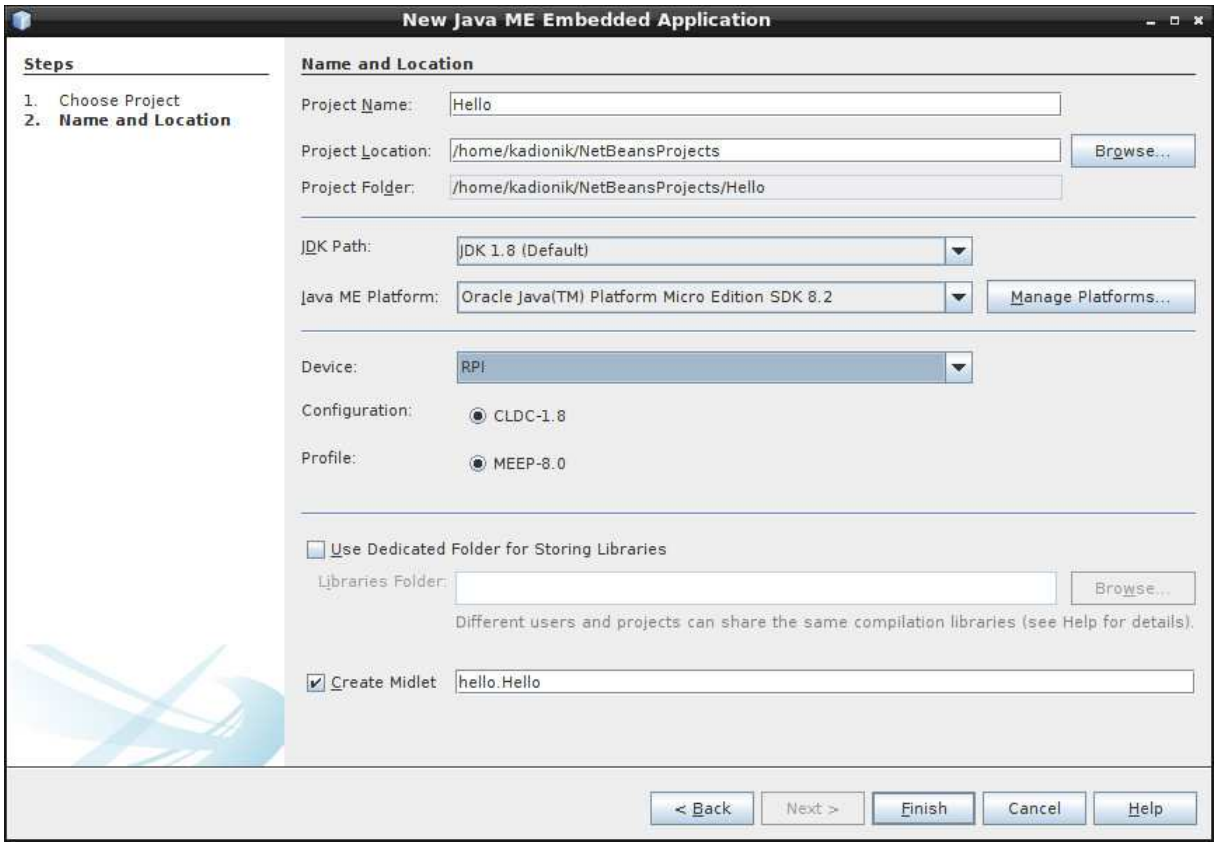
Cette opération n'est à faire qu'une seule fois.

On créera un nouveau projet NetBeans Java ME Embedded :



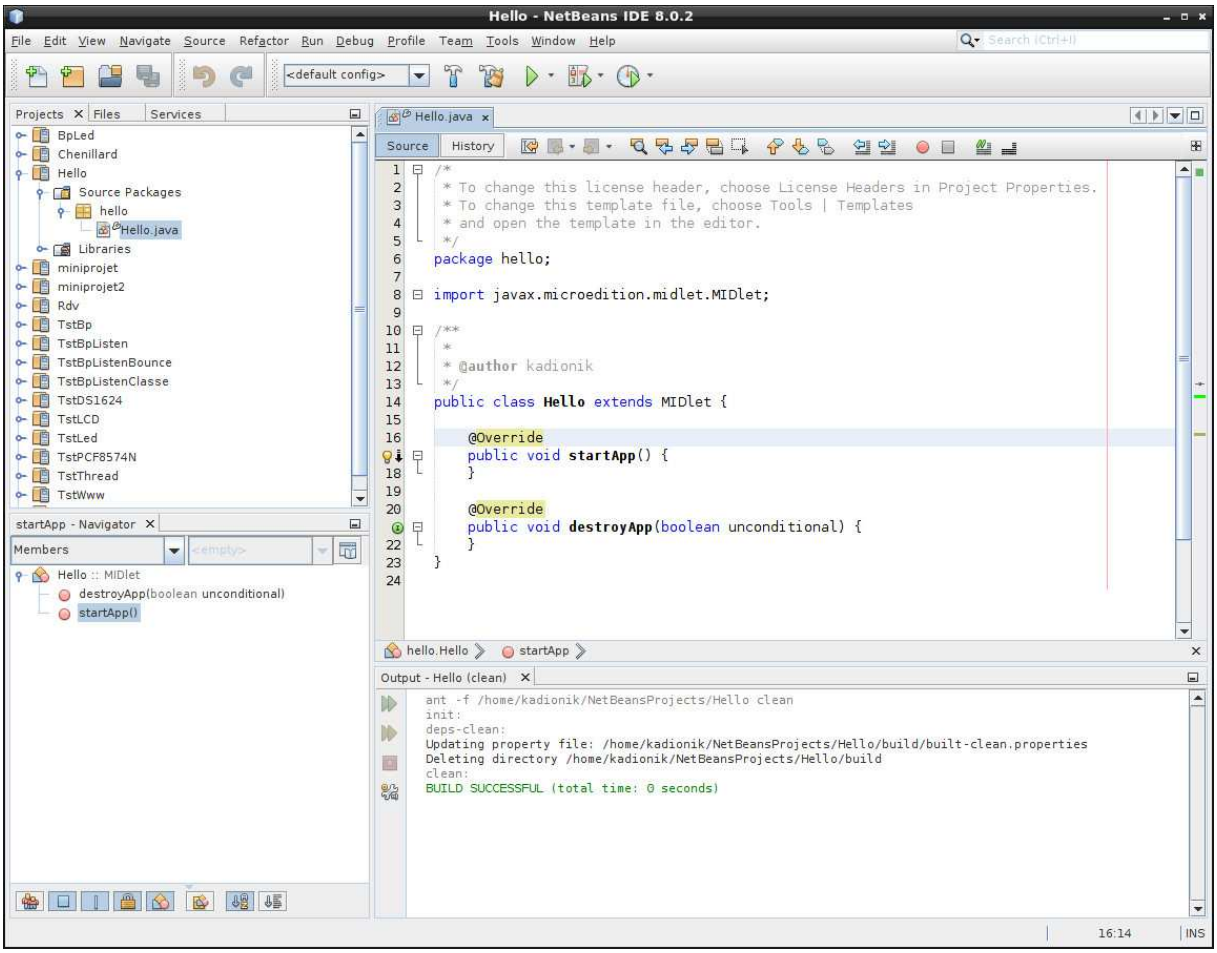
Création du projet `Hello` (1)

On appellera ce projet `Hello` et on choisira comme *device*, le *device* RPI. On laissera les autres options par défaut.



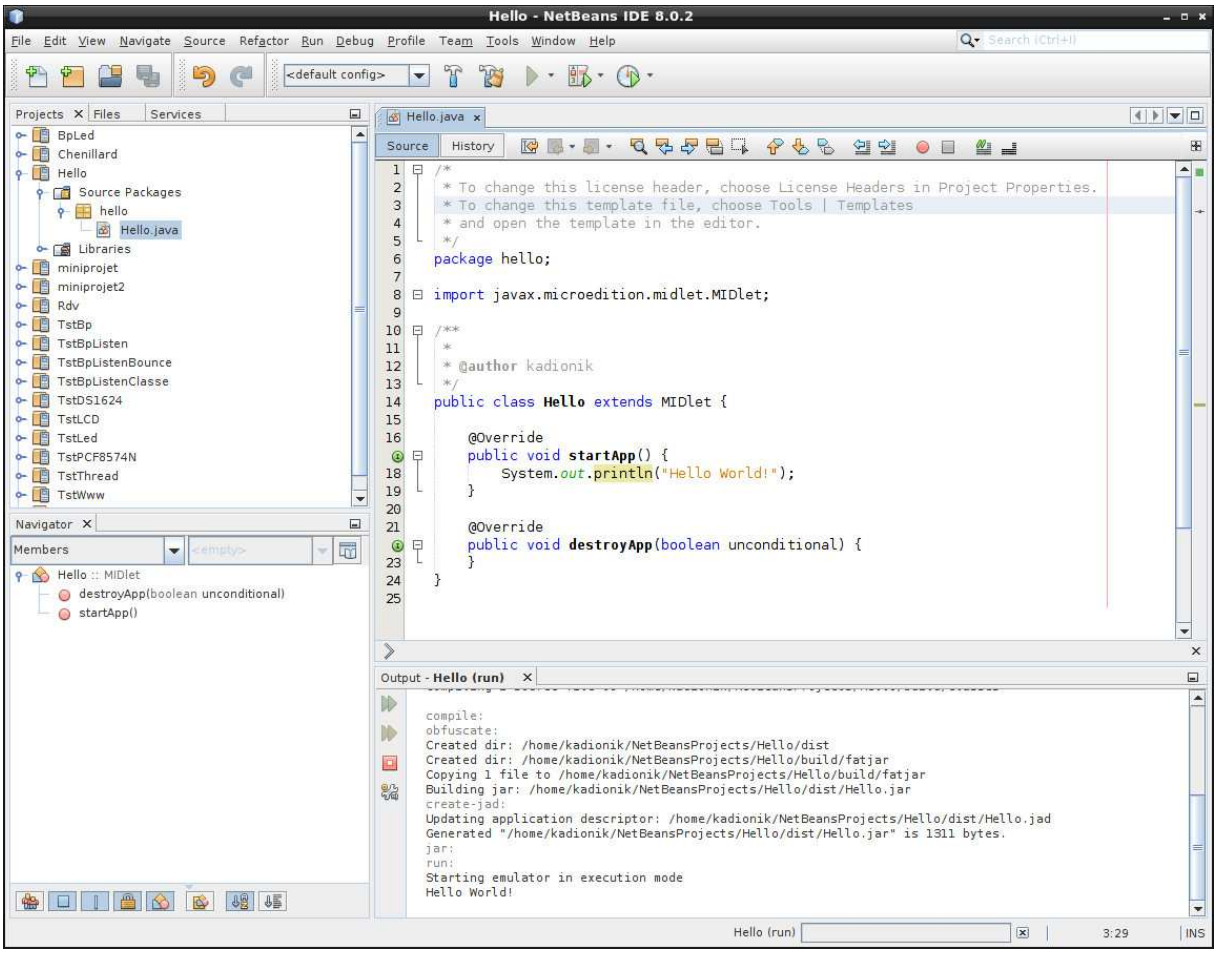
Création du projet `Hello` (2)

Le squelette du projet `Hello` est créé qu'il suffira de compléter dans le corps de la méthode `startApp()` de la *MIDlet*.



Création du squelette du projet `Hello`

On compilera puis lancera l'exécution de la *MIDlet* dans la carte cible rpi-java en cliquant sur l'icône en forme de triangle vert.



Exécution du projet `Hello`

On notera les traces d'exécution dans la zone d'exécution *output*.

La fenêtre RPI qui s'ouvre à l'exécution permet de voir l'état des *MIDlets* qui s'exécutent sur le *device* RPI.

Etat des *MIDlets* sur le *device* RPI

6. EX 2 : PROJET TSTLED

Le but de cet exercice est de faire clignoter au rythme d'une seconde la led LED1.

- Editer le projet `TstLed` et le modifier afin de réaliser l'exercice.
- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.
- On utilisera l'instruction Java :
  - `led1 = (GPIOPin) DeviceManager.open(LED1);`

  pour accéder à la led LED1.
- On utilisera l'instruction Java :
  - `Thread.sleep(1000);`

  pour attendre une seconde.

7. EX 3 : PROJET CHENILLARD

Le but de cet exercice est de réaliser un chenillard à la « K2000 » avec les leds LED1 à LED5.

- Editer le projet `Chenillard` et le modifier afin de réaliser l'exercice.
- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.
- On utilisera l'instruction Java :
  - `Thread.sleep(100);`

  pour une temporisation de 100 ms.

8. EX 4 : PROJET TSTBP

Le but de cet exercice est de détecter l'appui ou non sur les boutons poussoirs BP1 à BP4. Quand on a appuyé sur le bouton poussoir BPi (i de 1 à 4), on affichera le message « BPi PRESSED ».

- Editer le projet `TstBp` et le modifier afin de réaliser l'exercice.
- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.
- On utilisera l'instruction Java :
  ```
  bp1 = (GPIOPin)DeviceManager.open(BP1);
  ```
  pour accéder au bouton poussoir BP1 par exemple.
- On utilisera l'instruction Java :
  ```
  bp1.getValue();
  ```
  pour récupérer l'état courant du bouton poussoir BP1 (`false` = appuyé).

9. EX 5 : PROJET TSTBPLISTEN

Le but de cet exercice est de détecter l'appui sur le bouton poussoir BP1 en utilisant un *listener* Java. Il s'agit là de gérer l'action sur l'événement généré par l'appui sur le bouton poussoir BP1 contrairement à la scrutation active de l'exercice précédent.

- Editer le projet `TstBpListen` et le modifier afin de réaliser l'exercice. Quelle interface est implémentée ?

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

- On utilisera l'instruction Java :
```
bp1 = (GPIOPin) DeviceManager.open(new GPIOPinConfig(
                        0,
                        BP1,
                        GPIOPinConfig.DIR_INPUT_ONLY,
                        GPIOPinConfig.DEFAULT,
                        GPIOPinConfig.TRIGGER_FALLING_EDGE,
                        //GPIOPinConfig.TRIGGER_BOTH_EDGES,
                        false));
```
  pour accéder au bouton poussoir BP1.

- On utilisera l'instruction Java :
  `bp1.setInputListener(…);`
  pour installer le *listener*.

- On écrira la méthode Java :
  `public void valueChanged(PinEvent event) {}`
  pour préciser le code exécuté par le *listener* et lire l'état courant du bouton poussoir BP1.

10. EX 6 : PROJET TSTBPLISTENCLASSE

Le but de cet exercice est de détecter l'appui sur le bouton poussoir BP1 en utilisant un *listener* Java. On utilisera aussi le concept de classe Java. On créera une classe Java `BPDevice` qui gère un bouton poussoir qui sera passé comme paramètre lors de l'instanciation de la classe. On créera une classe `BPListener` qui implémente l'interface `PinListener` et qui implémente donc la méthode `valueChanged`. On testera le tout avec le bouton poussoir BP1.

- Editer le projet `TstBpListenClasse` et le modifier afin de réaliser l'exercice.

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

11. EX 7 : PROJET TSTLCD

Le but de cet exercice est de tester l'afficheur LCD en utilisant la classe `LCD2x16` qui est fournie. On affichera toutes les 100 ms sur la ligne 0, l'heure courante et sur la ligne 1 position 0 « LCD : i » et ligne 1 position 10 (i+2), `i` étant un compteur incrémenté à chaque cycle.
Un exemple d'affichage est donc :

```
12:34:55
LED: 10   12
```

- Editer le projet `TstLCD` et le modifier afin de réaliser l'exercice.

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

- On utilisera la méthode :
  ```
  Date now;
  now = new Date();
  getDateString(now));
  ```
  pour récupérer l'heure courante de la carte rpi-java.

- La classe `LCD2x16` fournit les méthodes suivantes :
  `init()` : initialisation de l'afficheur LCD
  `lcd_clear()` : effacement de l'afficheur
  `lcd_home()` : positionnement en ligne 0, position 0
  `lcd_line_puts(int line, String string)` : écriture de la chaîne de caractères `string` à la ligne `i` (0 ou 1)
  `lcd_pos(int li, int co)` : positionnement à la ligne `li` et position `co`

Un exemple d'usage de la classe `LCD2x16` est :

```
LCD2x16 display ;
display = new LCD2x16();

display.init();
display.lcd_clear();
display.lcd_home();
display.lcd_pos(1, 10);
display.lcd_puts("coucou");
```

12. EX 8 : PROJET TSTDS1624

Le but de cet exercice est de tester le capteur de température I2C DS1624 en utilisant la classe `DS1624` qui sera à écrire. On utilisera les méthodes I2C de base pour dialoguer avec le capteur de température. On affichera ainsi toutes les secondes la température du capteur.

On s'appuyera sur la documentation technique du capteur donnée en annexe 2.

La classe `DS1624` fournit les méthodes suivantes :

`start()` : lancement d'une acquisition de la température
`stop()` : fin de l'acquisition
`double read_temp()` : lecture de la température

Le capteur de température possède l'identificateur I2C numéro 0x48 sur le bus I2C numéro 1.

Le constructeur `DS1624()` de la classe `DS1624` crée un objet `config` puis ensuite un objet `slave` (le code source est donné). Les constantes et données importantes sont donc les suivantes :

```
final int LCD_I2C_ID = 0x48;
final int BUS_ID = 1;

final int DS1624_READ_TEMP = 0x??;
final int DS1624_START = 0x??;
final int DS1624_STOP = 0x??;

I2CDeviceConfig config;
I2CDevice slave;
```

Pour les méthodes `start()`, `stop()` et `read_temp()`, on utilisera alors les méthodes I2C suivantes :

`slave.write(commande)` : écriture I2C d'une commande.
`slave.read()` : lecture I2C d'une donnée 8bits.

- Editer le projet `TstDS1624` et le modifier afin de réaliser l'exercice.
- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

Un affichage avec un formatage d'un nombre réel sous la forme xy.z est donné par :
`System.out.format("%02.1f oC\n", temp)`

13. EX 9 : PROJET TSTTHREAD

Le but de cet exercice est de créer 2 *threads* Java `task1` et `task2` qui incrémentent chaque seconde une variable privée `i` et qui affichent sa valeur courante.

- Editer le projet `TstThread` et le modifier afin de réaliser l'exercice. Quelle interface est implémentée ?

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

- On créera 2 classes `task1` est `task2` qui implémentent l'interface `Runnable`, la méthode `run()` (boucle infinie du *thread*) et `stop()`.

Un exemple d'usage d'un *thread* est :

```
task1 t1;
Thread th1;

try {
            t1 = new task1();
            th1 = new Thread(t1);
            th1.start();
} catch (Exception ex) {
            ex.printStackTrace();
}

class task1 implements Runnable {
    private boolean quit = false;

    public void run() {
. . .
        try {
            while (!quit) {
. . .
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void quit() {

        quit = true;
    }
}
```

14. EX 10 : PROJET RDV

Le but de cet exercice est de découvrir le mécanisme de communication utilisée entre 2 *threads*.

- Editer le projet `Rdv` et comprendre ce qu'il réalise. Quel est le mécanisme de synchronisation implanté ? A quoi correspond la classe `Semaphore` ? A quoi correspond la méthode `P()` ? A quoi correspond la méthode `V()` ? A quoi sert la variable privée `counter` de la classe `Semaphore` ? A quoi sert le mot clé Java `synchronized` ? A quoi sert la méthode réservée Java `notify()` ? A quoi sert la méthode réservée Java `wait()` ?

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Aucune.

15. EX 11 : PROJET TSTWWW

Le but de cet exercice est de tester un mini serveur Web implémenté sous forme d'un *thread*.

- Editer le projet `TstWww` et comprendre ce qu'il réalise. On utilisera le navigateur Firefox sur le PC en forgeant l'adresse URL adéquate pour se connecter au serveur Web.

- A quoi correspond l'instruction Java :
```
ssc = (ServerSocketConnection) Connector.open("socket://:8080");
```

- A quoi correspond l'instruction Java :
```
sc = (SocketConnection) ssc.acceptAndOpen();
```

- A quoi correspondent les flux :
```
is = sc.openInputStream();
os = sc.openOutputStream();
```

- A quoi correspond la suite d'instructions Java suivantes sur le flux de sortie :
```
out.print("HTTP/1.1 200 OK\n");
out.print("Content-Type: text/html\n");
out.print("\n");
out.print("<H1><CENTER> Welcome to the rpi-java Web Server </CENTER></H2>");
out.print("Hello world!");
```

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

16. EX 12 : PROJET WWW

Le but de cet exercice est de créer un serveur Web qui renvoie l'heure courante et la température courante du capteur DS1624 de la carte rpi-java. La carte rpi-java simule ainsi un objet connecté que l'on interroge ici pour récupérer par le Web l'information issue d'un capteur…

- Editer le projet `Www` et le modifier afin de réaliser l'exercice.

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

- On pourra rajouter dans la réponse HTTP envoyée dans le flux de sortie vers le navigateur la balise HTML suivante pour forcer le rafraîchissement de la page toutes les secondes :
  `<meta http-equiv="Refresh" content="1">`
  soit l'instruction Java :
  `out.print("<meta http-equiv=\"Refresh\" content=\"1\">");`

17. EX 13 : MINIPROJET 1

Le but de cet exercice est d'utiliser toutes les E/S de la carte rpi-java. On affichera l'heure courante et la température courante sur l'afficheur LCD. L'appui sur le bouton poussoir BP1 affichera durant toute la durée de l'appui la température maximale enregistrée, l'appui sur le bouton poussoir BP2 affichera durant toute la durée de l'appui la température minimale enregistrée. Il y a enfin un chenillard sur les leds LED1 à LED5.

Un exemple d'affichage est donc :
```
12:34:55
TEMP=21.5 oC
```
L'appui sur BP1 affiche :
```
12:35:55
TMAX=25.7 oC
```
L'appui sur BP2 affiche :
```
12:38:22
TMIN=20.2 oC
```

- Editer le projet `Miniprojet` et le modifier afin de réaliser l'exercice.

- Télécharger dans la carte cible rpi-java et tester.

Indications :

- Vérifier que les permissions de la *MIDlet* sont correctes pour les E/S mises en œuvre dans l'exercice.

- On créera un *thread* `th_lcd` qui gérera l'affichage et donc l'afficheur LCD, le capteur de température et les boutons poussoirs BP1 et BP2.

- On créera un *thread* `th_led` qui gérera le chenillard sur les leds LED1 à LED5.

- On créera un *thread* `th_www` qui implantera un serveur Web qui renverra l'heure courante et la température courante comme pour le projet `www`.

- Attention, un seul *thread* gère les accès I2C ici `th_lcd`. Le *thread* `th_www` devra accéder à la température par variable partagée.

- On pourra utiliser l'objet `String` pour le formatage de l'affichage :
```
String d;
double temp;

d = String.format("%02.1f", temp);
```

18. EX 14 : MINIPROJET 2

Il s'agit de piloter les leds de la carte RPi via une application Linux qui sera un client TCP développé sur le PC hôte avec l'API *sockets* en C.

On créera une application Java ME qui sera le serveur TCP qui contrôlera les leds de la carte RPi.

Le protocole d'échange entre le client et le serveur peut être simple. On envoye 2 octets tels que :

- Caractère 1 : numéro de la led : de 1 à 5.
- Caractère 2 : état de la led : 0=éteinte, 1=allumée.

Le serveur renvoie le caractère 0 si OK et le caractère 1 si non OK.

Comme premier client TCP, on pourra utiliser la commande `telnet` pour tester le bon fonctionnement du serveur TCP.

Le canevas en langage C d'un client TCP est le suivant (les **???** sont à remplacer par les bonnes valeurs…) :

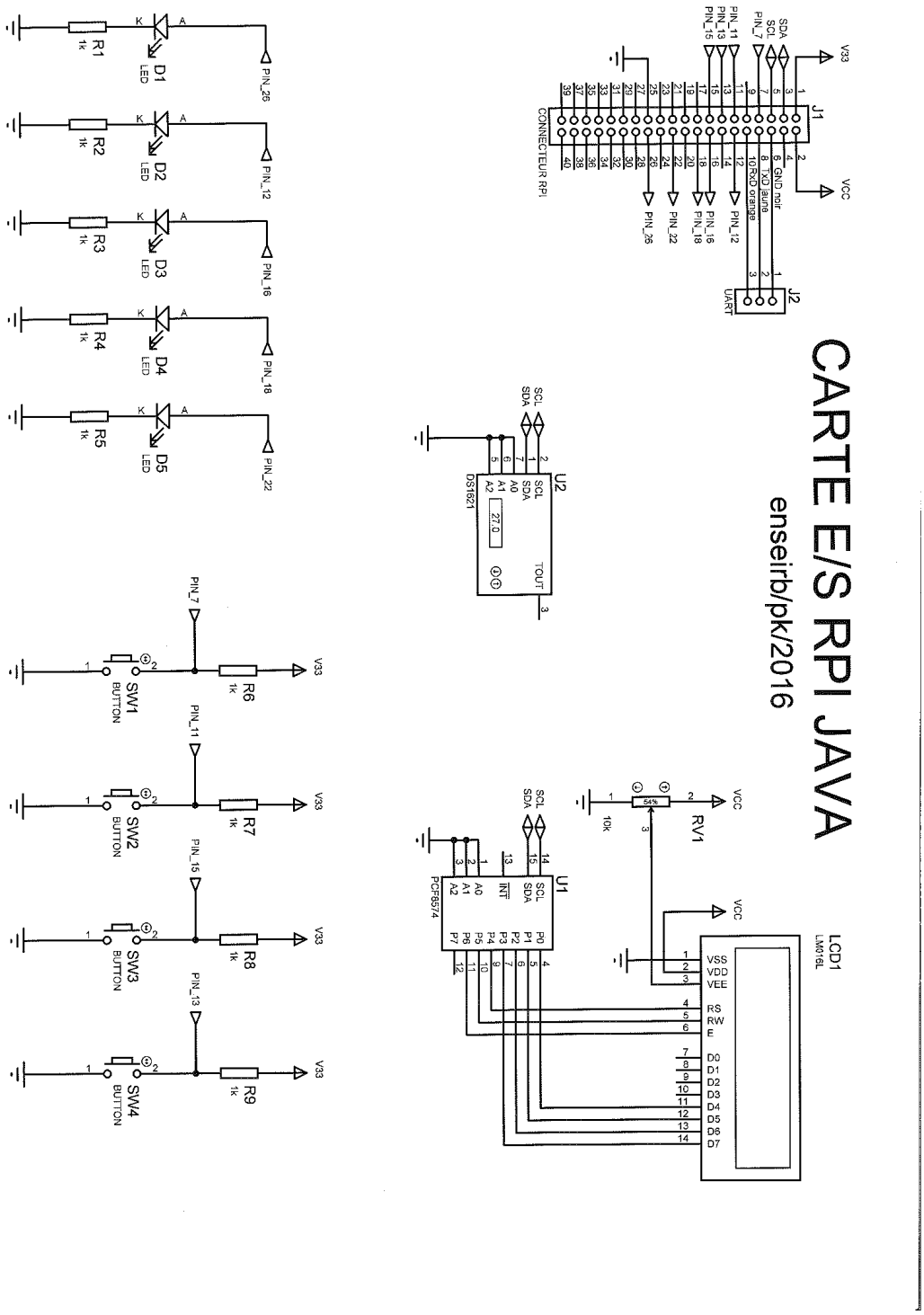
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(argc,argv)
     int argc ;
     char *argv[] ;

{
  int sd;
  struct sockaddr_in sa;             /* Structure Internet sockaddr_in */
  struct hostent *hptr ;             /* Infos sur le serveur */

  char *serveur ;                    /* Nom du serveur distant */
  int port;

  /* verification du nombre d'arguments de la ligne de commande */
  if (argc != 3) {
    printf("myclientP. Erreur d'arguments\n");
    printf("Syntaxe : %% myclient nom_serveur numero_port\n");
    exit(1);
  }

  /* Recuperation nom du serveur */
  serveur = argv[1];

  /* Recuperation numero de port */
  port = atoi(argv[2]);

  /* Recuperation des infos sur le serveur dans /etc/hosts pour par DNS */
  if((hptr = gethostbyname(serveur)) == NULL) {
    printf("Probleme de recuperation d'infos sur le serveur\n");
    exit(1);
  }
```

```
  /* Initialisation la structure sockaddr sa avec les infos  formattees : */
  /* bcopy(void *source, void *destination, size_t taille);               */
  bcopy((char *)hptr->h_addr, (char*)&sa.sin_addr, hptr->h_length);

  /* Famille d'adresse : AF_INET ici */
  sa.sin_family = hptr->h_addrtype;


  /* Initialisation du numero du port */
  sa.sin_port = htons(port);

  sd = socket(AF_???, ???, 0);
  connect(sd, (struct sockaddr *) &sa, sizeof(sa));
  . . .
  close(sd);
  exit(0);
}
```

19. REFERENCES

- Site de NetBeans : https://netbeans.org/
- Site de J2ME : http://www.oracle.com/technetwork/java/embedded/javame/index.html

20. ANNEXE 1 : SCHEMA ELECTRONIQUE DE LA CARTE D'E/S DE LA CARTE CIBLE RPI-JAVA



21. ANNEXE 2 : DOCUMENTATION SUR LE CAPTEUR I2C DS1624

22. ANNEXE 3 : CONFIGURATION RESEAU HOTES ET CIBLES

	POSTE PC01	
	NOM	**ADRESSE IP**
HOTE	citron07	192.168.4.7
CIBLE	rpi01	192.168.4.101

	POSTE PC02	
	NOM	**ADRESSE IP**
HOTE	citron08	192.168.4.8
CIBLE	rpi02	192.168.4.102

	POSTE PC03	
	NOM	**ADRESSE IP**
HOTE	citron09	192.168.4.9
CIBLE	rpi03	192.168.4.103

	POSTE PC04	
	NOM	**ADRESSE IP**
HOTE	citron10	192.168.4.10
CIBLE	rpi04	192.168.4.104

	POSTE PC05	
	NOM	**ADRESSE IP**
HOTE	citron11	192.168.4.11
CIBLE	rpi05	192.168.4.105

	POSTE PC06	
	NOM	**ADRESSE IP**
HOTE	citron12	192.168.4.12
CIBLE	rpi06	192.168.4.106

Masque de sous réseau : **255.255.255.0**

Exemple : configuration réseau de la carte cible `rpi01` :

```
target# ifconfig eth0 192.168.4.101
```