Code Guidelines ¶

Author: Vincent Petrella

Here are the global guidelines for writing our code:

Stuff you already know:

- One class per file.
- Classes should have the exact same name as (*.java) filename they are in, Case Sensitive.

Guidelines:

Pascal casing: The first character of each word should be in Upper Case, others in Lower Case:
"PascalCasingExample, Blah, FooBar."

Camel casing: The first character of each world should be in Upper Case, **except for the first word**, the others are in Lower Case:
"thisIsCamelCasing, toto, fooBar"

Classes and methods: Use Pascal Casing!

```
public class HelloWorld  // Pascal casing pour le nom des classes
{
   void SayHello(string name)  // Pascal casing pour le nom des méthodes
   {
   }
}
```

Variables and parameters: Use Camel Casing!

```
public class HelloWorld
{
   int totalCount = 0;  // Camel casing pour les variables...
   void SayHello(string nameOfCat)  // ... et les paramètres des méthodes
   {
   }
}
```

No other notation (hungarian etc...)

```
String m_sName; // Non
int nAge;       // Non
```

Do Not use underscore "_" in your variable names... "Pitié!"

```
String _name;           // Non
Vector3 position_du_bisounours;      // Non
```

Use meaningful names for your variables
Do not use abbreviations:

```
Vector2 position, acceleration, forceDuProut;          // Oui
Vector2 pos, acc, prout;                               // Non
```

Exception for your while/for loops!

```
for ( int i = 0; i < count; i++ ) // OK
{
                        //...
}
```

Jump to a newline to place your bracket after an "if", "for" statement (for readability). If an "if" block is only 1 instruction, you can omit the brackets, BUT DON'T MIX STYLES!

```
if ( ... )
{
   CallFunction(); //Yes.
}
else
{
   AnotherFunction(); //Yes! I love you, seriously.
}

if ( ... )
   CallFunction();
else ( ... )
   AnotherFunction(); // Yes, you can do that...

if ( ... ) {
   CallFunction();       // Please go to a newline for "{".
}

if ( ... )
   CallFunction();
else
{
   // There we go, you mixed styles and God just killed a kitty cat.
   AnotherFunction();
}
```

If a function is only 1 line long (Getter / Setter), you can write it on a single line:

```
public void SetSpeed(float speed)   {       speed = speed };
public float GetSpeed()             {       return speed  };
```

Try to use a single space before and after a parenthesis, as well as before and after a comparison, to aerate the code:

```
if ( showResult == true )
{
   for ( int i = 0; i < 10; i++ )
   {
      // OK
   }
}
```

To imply that a method exists to receive a message or an event, we use **On**Foobar():

```
void OnTravel(Vector2 traveLocation) { ... }
void OnStop() { ... }
```

That brings us to an important part of these code Guidelines: Using threads.

Threads:

Our Robot is a "complex" system, and thus is composed of several subsystems doing their jobs separately, and periodically. We will most likely implement theses threads as "TimerListener" for the final system in order to properly schedule executions, but the principle remains the same.

Now a thread is something we said that is executed periodically, until the thread itselft is terminated. Based on the state of the said subsystem, the thread will execute different code, relative to the situation.

For example, in Lab3, we used Threads for our Navigation and could derive several states: "iAmAtDesiredDestination", "iAmTravellingToDestination", "iAmAvoidingAnObstacle".

In each different state, the behavior of the system is expected to be different, and thus the code executed is also different:

Typical thread loop:

```
void run()      //or timedOut() for TimerListener
{
        while( !terminationRequest )
        {
            if( stateOne )
            {
                DoBehaviorOne();
            }
            else if( stateTwo )
            {
                DoBehaviorTwo();
            }
            else
            {
                DoDefaultBehavior();
            }

        }
}
```

These states are supposed to be "toggleable" by the class itself, and most importantly by other classes which would manage the thread. That's where OnFoobar() methods come in handy.
In Lab3, we were given an array of coordinates our robot had to travel to successively. These coordinates were defined in the Lab3 class. We then used a method called:

```
void OnTravelTo(int[] Coordinate)
{
```

which sets the state of Navigation to:

```
        this.isNavigating = true;
        this.isTravelling = true;
```

and then sets the variables used by the Travelling() method to head to the desired point:

```
        this.desiredAngle = ComputeDesiredAngle(desiredCoord, currentHeading);
        ...
}
```

Understand that **OnTravelTo()** serves as a trigger to toggle the isTravelling behavior, until **Travelling()** realizes it arrived to the right destination and calls **OnNavigationStop()** to set isNavigating and isTravelling to false.

Then the thread loop for our navigation would look like:

```
void run()
	{
		while ( true )  // We don't expect the thread to be terminated separately.
		{
			if ( isNavigating )
			{
				if ( isTravelling )
				{
					if ( GetDistanceToWall() < criticalDistance )
					{
						OnObstacleAhead();
					}
					else
					{
						Travelling();
					}
				}

				if ( isAvoidingObstacle )
				{
					ObstacleFollower();
				}
			}
		}
	}
```

Where:

- **GetDistanceToWall()** returns the distance given by ultrasonic sensor polling thread.
- **OnObstacleAhead()** toggles isAvoidingObstacle behavior which is implemented by **ObstacleFollower**(). (Also sets isTravelling to false)
- **Travelling()** implements the travelling behavior (tracking the distance left to destination and desired heading), and sends a **OnNavigationStop()** to the class when desiredLocation ~= currentLocation.

Then when isNavigating == false, the navigation thread is just waiting for another **OnTravelTo()** event.

That's it for thread handling, note that a TimerListener works exactly the same but doesn't have a thread loop, because timedOut() will be called each time the timer sends a signal.

As always, if you have any question/suggestion about writing code, just ask, we still have a couple days before actually start coding :).