GRAPHIC LIBRARIES MANAGEMENT : Interface IWindow

The `IWindow` interface is used as a base for all our graphic libraries. It is imperative to use it if you want to implement a new graphic library that is compatible with our program.

Graphic libraries that you can implement:

The arcade being an old machine, it is only compatible with a few graphic libraries. Here is a list of compatible libraries :

- NDK++: `arcade_ndk++.so`
- aa-lib: `arcade_aalib.so`
- libcaca: `arcade_libcaca.so`
- Allegro5: `arcade_allegro5.so`
- Xlib: `arcade_xlib.so`
- GTK+: `arcade_gtk+.so`
- SFML: `arcade_sfml.so`
- Irrlicht: `arcade_irrlicht.so`
- OpenGL: `arcade_opengl.so`
- Vulkan: `arcade_vulkan.so`
- Qt5: `arcade_qt5.so`

Be careful when creating the library to use the correct name, only the bellow formats are supported. Moreover, when your dynamic library is created, you must place it in the `./lib` folder for it to be used by the program. If you don't, it won't be used by our arcade.

HOW TO : add a new graphic library

All the graphic libraries loaded in the program must be dynamic.
For our project, your graphic library must implement this function for its creation:

```
extern "C" std::unique_ptr<IWindow> createLib();
```

It will return a pointer to the newly created class that inherits from the `IWindow` interface. Here is an example of implementation for the NCurse library, that you can use as a template for your own implementation :

```
extern "C" std::unique_ptr<IWindow> createLib()
{
    return (std::make_unique<NcWindow>());
}
```

/!\ We are storing a `unique_ptr` of our window in the `Core` class, so please don't use `shared_ptr` or any of other type of pointer !

In order for your graphic library to be compatible with our program,you must firstly create your own class that inherits from the `IWindow` interface. Several methods are expected inside of it:

- display();
- clear();
- pollEvent(Events&);
- setTitle(const std::string&);
- setSize(const vec2int&);
- draw(const Line&);
- draw(const Rectangle&);
- draw(const Point&);
- draw(const Text&);
- play(const ASound&);
- setFramerate(int framerate) noexcept;
- getStatus();

General methods:

```
void display()
```

- Display all previously drawn elements on the window.

```
void clear()
```

- Clear the previous draw.

```
bool pollEvent(Events& event)
```

- Pop the event on top of the event queue.

- return if the queue is empty or not.

```
void setTitle(const std::string& title)
```

- Sets the window title with the new title taken in parameter.

```
void setSize(const vec2int& size)
```

- Sets the window size with the new size taken in parameter.
- The `vec2int` structure contains a pair of int `x` and `y`.

```
Status getStatus() noexcept
```

- Return `Status`, an enum representing the current status of the window to the `Core`.
- Its return value is then used for event handling in the main loop of the project.
- You can find more informations about the `Status` enum in the `Utils.hpp` file.

```
void setFramerate(int framerate) noexcept
```

- Sets the framerate attribute of the class inherited from the `IWindow` interface with the new framerate taken in parameter.

Draw methods:

These are all the methods used to draw an element in the program. They all take a reference of a class herited from the `AShape` abstract class. For more details about this class and how it stores informations, please refer to the `AShape.pdf` file. For each class that herited from the `AShape` class, you must implement a `draw` method in your graphic library.

```
void draw(const Line& line)
```

- Draw a line.

```
void draw(const Point& point)
```

- Draw a point.

```
void draw(const Rectangle& rectangle)
```

- Draw a rectangle.

```
void draw(const Text& text)
```

- Draw text.

```
void play(const ASound& sound)
```

- Play a sound.