DEVELOPPER DES PLUGINS BUKKIT !

Bonjour à tous !
Aujourd'hui, je souhaiterai vous faire partager mon savoir et ma connaissance à travers un tutoriel pour que vous puissiez vous même développer VOS propres plugins pour Bukkit !

Prérequis :

● **Java** installé sur votre ordinateur.
● **Minecraft**.
● Un serveur de test sous **CraftBukkit** / **Spigot** (créez simplement un serveur local).
● Un **IDE** pour **Java** (personnellement, j'utilise **Eclipse** mais libre à vous d'utiliser celui que vous voulez !).
● Des notions en **Java** (surtout pour la partie sur le **Scheduler**).

Veuillez noter que ce tutoriel est constamment mis à jour.
De plus, le code disponible ici est présenté de manière aéré pour un apprentissage plus performant.

I – Apprendre à utiliser la JavaDoc de Bukkit

Alors tout d'abord, qu'est-ce-qu'une **JavaDoc** ?
Les gens ayant des notions en Java le savent déjà :P

Une **JavaDoc**, c'est tout simplement un outil **INDISPENSABLE** pour pouvoir développer avec une API ! (Bukkit en possède une, comme toute API qui se respecte, j'y reviendrais plus tard !).

Alors comment l'utiliser ?
C'est très simple et vous allez voir, une fois qu'on y a goûté, on ne peux tout simplement plus s'en passer !

Déjà, voici le lien des **JavaDoc** de Bukkit.
Comme vous le voyez, vous avez trois choix avec des numéros de versions. Vous prenez le numéro de la version de l'API Bukkit que vous possédez.

A l'heure ou j'écris ce tutoriel, la version recommandée 1.6.2 n'est toujours pas sortie, nous allons donc nous diriger vers la **JavaDoc** de la version **Beta** 1.6.2. Vous pouvez prendre la version *classique* ou la version *Doxygen*. Personnellement, je vous conseille largement d'utiliser cette dernière, après cela ne regarde que moi ! (Cette partie est exclusivement consacrée à la version *Doxygen*, désolé pour les autres ^^).

Donc vous arrivez sur une belle page avec beaucoup de bleu :P
En haut à droite, vous avez une barre de recherche. Cette barre : c'est le **TOUT** !
Il vous suffit d'entrer le nom d'un objet, d'une fonction et PAF ! La **JavaDoc** vous donne une documentation complète de ce que vous recherchez.

Par exemple, imaginons que je cherche une fonction qui me permette d'augmenter / de diminuer la vie maximale d'un joueur.

Pour cela j'entre le nom de l'objet « **Player** » dans la **JavaDoc** et on attend quelques secondes.
Ensuite on sélectionne l'objet et il ne reste plus qu'à chercher la fonction souhaitée dans la page qui s'ouvre à vous !
Donc un petit CTRL + F et on entre le mot clé : « **Health** » (*Santé* en anglais) et là on tombe sur...

Oh my god ! **49 occurrences** ^^

Il suffit d'affiner notre recherche, pour cela se poser les questions :

Je cherche quoi ? Quels sont les mots pour décrire ce que je cherche ?

Dans cet exemple, je cherche à augmenter / diminuer la vie maximale d'un joueur.

Donc, vue que on ne peut pas retirer de vie (un petit CTRL + F « **remove** » vous le prouve...) et que je veux une seule et même fonction pour la retirer et l'ajouter, il va falloir la **mettre**.

Et pour les connaisseurs du Java (même pour ceux qui ne parle ne serait-ce qu'une once d'anglais...), ce mot clé s'appelle **set**.

Nous savons déjà que la fonction contiendra « set » et « health ». Et à votre avis, comment on dit « maximum » en anglais ? Je ne vais pas vous faire mariner plus longtemps et comme je pense que vous avez tous trouvé la réponse, la fonction est...
setMaxHealth(...) !

Ah oui mais quand on recherche « **setMaxHealth** » dans la doc de l'objet « **Player** », ça ne nous donne rien...

Ah c'est embêtant... Eh bien, vous savez pourquoi ? C'est parce que les fonctions que vous voyez sur cette page ne sont applicables uniquement qu'à l'objet **Player**... Et **setMaxHealth(...)** est applicable aux cochons, poules et moutons aussi ! (Notion de **POO**.)

Eh bien alors comment trouver la documentation pour cette fonction ?

La réponse est toute bête... Vous placez votre curseur dans la barre de recherche et vous tapez « **setMaxHealth** », un petit clique et... TADAAAAM ! Vous apprenez tout ce que vous voulez savoir sur cette fonction, sa description, ses arguments, etc...

Bon c'est pas tout ça mais on est pas là pour ça nous, on est là pour développer ! :P

II – L'API de Bukkit

Après ce petit chapitre sur le fonctionnement de la **JavaDoc** de Bukkit, nous pouvons enfin commencer à développer !

Tout d'abord, pour pouvoir créer un plugin, il vous faut télécharger l'**API de CraftBukkit** (autrement appelée **Bukkit**). Pour cela, rendez-vous sur cette page. Je vous conseille de prendre les **Beta Builds** au minimum (c'est à dire, pas les versions en développement sauf si vous voulez être sûr que votre plugin marchera sur une future version !).

Je vous conseille de placer le fichier JAR téléchargé dans un répertoire « **libs** » sur votre disque dur et l'appeler « **Bukkit.jar** », cela facilitera la migration vers les prochaine versions car vous n'aurez juste qu'à télécharger la nouvelle version et remplacer l'ancienne. (Encore une fois cela ne regarde que moi ^^)

Une fois en possession du fichier JAR, ouvrez **Eclipse**.

Dans ce chapitre nous allons créer un plugin qui porte le doux nom de « **HiMinecraft** » qui dit « **Bonjour !** » à la console lorsqu'il est démarré et que lorsque vous tapez la commande « /**hi** » envoi un message au joueur disant « **Salut !** ». Cette commande nécessitera également la permission **himinecraft.say** et broadcastera un message en vert sur le serveur disant « ***NomDuJoueur* vous dit bonjour à tous !** ».

Sans plus tarder, commençons... :P

Nous allons créer un nouveau projet Java. Pour cela *File → New → Java Project*.
Nous allons nommer notre projet « **HiMinecraft** », mais libre à vous de mettre n'importe quel nom ici, cela n'aura aucun impact dans le jeu.

La première des choses à faire est d'ajouter Bukkit (et sa **JavaDoc**) au **Build Path**. Pour cela : *Clic droit sur votre projet → Properties → Java Build Path →* Onglet *Librairies → Add Externals JARs*... Et à partir d'ici, vous sélectionnez le JAR précédemment téléchargé (une façon de vous renvoyer en arrière si vous n'avez rien lu :P).

La deuxième chose à faire est de créer un fichier appelé « **plugin.yml** » dans votre projet (*Clic droit sur le projet → File → New → File*). Voici une documentation française très claire sur le plugin.yml, je mets la solution ici uniquement à titre d'exemple :P

Donc, une fois que vous avez lu la documentation ci-dessus, voici notre **plugin.yml** :

```
name: HiMinecraft
main: com.skyost.himinecraft.HiMinecraftPlugin
description: Soyez polis :P
version: 0.1
author: Skyost
website: http://www.skyost.eu/
commands:
 hi:
  aliases: hello # Tapper /hello aura le meme effet que /hi.
  description: Rien a apprendre, tout a comprendre ! # Comme vous le voyez, les
accents et autres caracteres sont interdits !
  usage: Utilisez simplement /hi depuis le jeu !
  permission: himinecraft.say
permissions:
 himinecraft.say: # Vous pouvez egalement declarer les permissions directement
dans le code ce qui vous permet un message personnalise avec
player.hasPermission(...) !
  default: true # Permet de specifier la valeur par defaut de la permission.
Utile quand on a pas de plugin de permissions.
  description: Vous autorise a utiliser /say.
```

Voilà voilà, maintenant créez une nouvelle classe (*clic droit sur le dossier source → New → Class*) et dans **package**, vous mettez **EXACTEMENT** ce qui se trouve dans le « **main** » du **plugin.yml**, sauf avant le dernier point ! (Dans ce cas ci, « **com.skyost.himinecraft** », cela représente le package de la classe principale.) Et dans **Name** vous mettez **EXACTEMENT** tout ce qui se trouve après le dernier point ! (Ici, « **HiMinecraftPlugin** » qui représente la classe principale du plugin).

Bien, votre classe étant créée et ouverte, nous pouvons (vraiment) commencer ! :P

Tout d'abord, après le « **public class HiMinecraftPlugin** » vous tapez « **extends JavaPlugin** », les gens ayant des notions en Java savent à quoi cela correspond, je ne reviendrai donc pas là dessus.

Nous allons commencer par faire dire un message à la console lorsque le plugin démarre, pour cela, il vous suffit de rajouter le code suivant :

```
@Override
public void onEnable() {

}
```

Ce code sera activé lorsque le plugin est activé, donc d'après vous, comment peux-t-on imprimer un message sur la console ? Et bien oui, avec un bête **System.out.println(...)** !

Ce qui nous donne :

```
@Override
public void onEnable() {
	System.out.println("Bonjour !");
}
```

A titre indicatif, **Bukkit.getConsoleSender()** renvoi un object **« ConsoleCommandSender »** ;)

Voilà, vous pouvez compiler et tester le code (*Sélectionnez votre projet → File → Export... → JAR File → Cochez votre projet,* et sélectionnez l'option *Export generated class files and ressources* et si vous le souhaitez *Compress the contents of the JAR file,* indiquez le chemin que vous souhaitez, je vous conseille le dossier « **plugins** » de votre serveur de test → *Next → Next →* Vous générez un *Manifest* et *Finish !*).

Démarrez votre serveur et...

```
[INFO] Bonjour !
```

Comme vous le voyez, cela marche plutôt bien ^^
Bon... Nous allons continuer avec les commandes maintenant !
Je vous laisse étudier ce bout de code et nous passerons aux explications :

```
@Override
public boolean onCommand(final CommandSender sender, Command cmd, String label,
String[] args) {
	Player player = null;

	if(sender instanceof Player) {
		player = (Player)sender;
	}
	else {
		return false;
	}

	if(cmd.getName().equalsIgnoreCase("hi")) {

	}

	return true;
}
```

Ce code est passe partout, vous pouvez vous en servir dans tout vos plugins qui utilisent des commandes.
Donc, déjà la question que vous vous posez tous est : Pourquoi un **boolean** ? (Comment ça non ? :P)
Tout simplement pour voir si la commande est correctement exécutée ou non !
Dans cette exemple (au début du code), nous vérifions si le **CommandSender** et un joueur, si ce n'en est pas un, nous retournons false. Cela enverra le message décrit dans le *usage* dans le plugin.yml. (Ici, cela enverra « Utilisez simplement /hi depuis le jeu ! ».) Une petite précision toutefois : on peut considérer à votre niveau que les **CommandBlocks** sont traités comme la console ;)
Donc, nous voulons que cette commande envoi « Salut ! » au joueur. Je vous laisse réfléchir quelques instants avant de vous donner le code...

Ça y est ? Voici le code :

```
player.sendMessage("Salut !");
```

Eh oui, ce n'est pas pour rien que j'ai fais un chapitre sur la JavaDoc quand même ^^
Si vous ne savez pas quel fonction utiliser, n'hésitez pas à faire un tour sur la JavaDoc ! Ou alors sur les forums de Bukkit.org, eux aussi très complet !

Maintenant comment diffuser un message sur le serveur ?
La classe **JavaPlugin** à un fonction très intéressante qui est : **getServer()** et qui renvoi un objet **Server**. Un petit coup de **JavaDoc** et le tour est joué ! (Vous verrez plus tard que l'on a pas forcément besoin de cet objet **Server** ;))

```
this.getServer().broadcastMessage(ChatColor.GREEN + player + " vous dit bonjour à tous !");
```

Osez me dire que ce code est bon et je vous balance une boite de Pringles !

NON ! CE N'EST PAS CORRECT !

Qu'est-ce-que la variable **player** ? C'est un objet **Player**, et non pas un **String** !
Donc, pour obtenir le nom du joueur, il faut faire tout simplement :

```
this.getServer().broadcastMessage(ChatColor.GREEN + player.getName() + " vous dit bonjour à tous !");
```

Voilà qui est déjà mieux...
Vous pouvez maintenant tester !

Tout marche comme sur des roulettes et vous avez terminé :D
Maintenant, nous pouvons voir deux ou trois petites choses pour améliorer le plugin...

III – Les événements

C'est quoi un événement ?
Un événement est un fonction qui est appelée lorsque quelque chose se produit, par exemple lorsqu'un joueur se connecte, lorsqu'il chat, lorqu'il ramasse un item au sol, etc... Vous pouvez accéder à la liste complète des événements en tapant « **Event** » dans la JavaDoc **Doxygen** de Bukkit.

Donc, je veux que lorsqu'un joueur se connecte, le message de connexion soit : « ***NomDuJoueur* vient de se connecter, soyez polis avec lui :3** » et de couleur turquoise. Alors, comment faire ?

Voici ma solution pour appeler l'événement :

Créez une classe appelée **Listeners**, elle implémentera **Listener** (pour cela, rajoutez « **implements Listener** » juste après le « **public class Listeners** »).

Dans cette classe, ajoutez ce code :

```
@EventHandler
private void onPlayerJoin(PlayerJoinEvent event) {

}
```

Vous avez remarqué le **@EventHandler** ? Si vous ne l'ajoutez pas, votre événement ne sera tout simplement pas appelé.
Une deuxième chose : vous devez obligatoirement enregistrer vos événements ! Pour cela, rien de plus simple, ajoutez simplement le code suivant à votre **onEnable()** :

```
this.getServer().getPluginManager().registerEvents(new Listeners(), this);
```

Si vous avez bien fait attention, et que vous regardez la JavaDoc, vous comprendrez ce code ;)
(Encore une fois ici, on pourra remplacer l'objet **Server**, nous ferons cela dans un autre chapitre !)

Bon bien, maintenant, comment changer le message de connexion ?
Utilisez la fonction **setJoinMessage(...)** de l'event **PlayerJoinEvent** comme ça :

```
@EventHandler
private void onPlayerJoin(PlayerJoinEvent event) {
      event.setJoinMessage(ChatColor.AQUA + event.getPlayer().getName() + "
vient de se connecter, soyez polis avec lui :3");
}
```

Alors, vous comprenez ?
Exécutez le code et... TADAAAAAAAAAAAAAAAAAM !!!

Vous êtes fier de vous ?
VOUS AVEZ CRÉÉ VOTRE PREMIER PLUGIN BUKKIT !! :D

IV – La configuration

Eh oui mais non, c'est pas (encore) fini !
Il nous reste une autre chose à voir : **la configuration de vos plugins !**

Alors personnellement, je vous conseille une **librairie** très facile à prendre en main et qui permet de sauvegarder tout et n'importe quoi (y comprit la couleur pour les messages dont je ne suis pas sûr que l'on puisse la sauvegarder avec l'API Bukkit...) : Skyoconfig. Un tutoriel est disponible ici. En plus, elle est faîte par moi :D

Je ne l'aborderai malheureusement pas dans ce tutoriel, je préfère vous faire toucher aux configurations incluses par défaut dans l'API de Bukkit au lieu de vous donner une librarie tout de suite.

Bien, alors maintenant, nous allons voir comment utiliser la configuration.

Créez un nouveau fichier appelé « **config.yml** » dans votre projet.
Ce fichier contiendra les lignes suivantes :

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
#                                                           #
#              Ceci est un commentaire !                    #
#                                                           #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

messages:
  '1': Bonjour !
  '2': Salut !
  '3': /player/ vous dit bonjour !
  '4': /player/ vient de se connecter, soyez polis avec lui :3
```

Maintenant, ajoutez cette fonction à la classe principale de votre plugin :

```
public void loadConfig() {
      this.getConfig().options().copyDefaults(true);
      this.saveConfig();
}
```

La première ligne vous permet de copier le fichier config.yml si il est présent dans le fichier JAR. (Pour cela, n'oubliez pas de le cocher lors de l'export de votre fichier JAR.) Dans notre cas, il faut absolument qu'il soit présent car nous n'utilisons pas la fonction **addDefault(...)** ! (cf. **JavaDoc** sur **FileConfiguration**.) Et la seconde sauvegarde toutes les options.

A partir de cela, je vais pouvoir vous montrer comment charger nos messages.
Allons dans notre **onEnable()** et remplaçons cette ligne :

```
System.out.println("Bonjour !");
```

Par celle-ci :

```
System.out.println(this.getConfig().getString("messages.1"));
```

Je vous vois sourire derrière votre écran... Vous avez compris ? :D

Et oui, le **getString(...)** renvoi un objet **String** se trouvant dans la config, par exemple si le message à aller chercher est de cette forme :

```
Mes-objects:
  'string-1': Bonjour !
```

Vous utiliseriez donc :

```
System.out.println(this.getConfig().getString("mes-objets.string-1"));
```

Pas bien compliqué hein ?
Alors je vous laisse faire la suite avant de vous livrer les autres messages... :P

Ça y est, c'est bon ?
Voici le code de la classe **HiMinecraftPlugin** :

```
public class HiMinecraftPlugin extends JavaPlugin {

	@Override
	public void onEnable() {
		this.getServer().getPluginManager().registerEvents(new
Listeners(this), this); // Laissez le this, on s'en servira après !
		System.out.println(this.getConfig().getString("messages.1"));
	}

	public void loadConfig() {
		this.getConfig().options().copyDefaults(true);
		this.saveConfig();
	}

	@Override
	public boolean onCommand(final CommandSender sender, Command cmd, String
label, String[] args) {
		Player player = null;

		if(sender instanceof Player) {
			player = (Player)sender;
		}
		else {
			return false;
		}

		if(cmd.getName().equalsIgnoreCase("hi")) {
			player.sendMessage(this.getConfig().getString("messages.2"));
			String message =
this.getConfig().getString("messages.3").replace("/player/", player.getName());
// Notez bien le replace(), il n'y pas de fonction toute près faite pour
désigner un joueur !
			this.getServer().broadcastMessage(ChatColor.GREEN + message);
		}

		return true;
	}

}
```

Et voici le code de **Listeners** :

```
public class Listeners implements Listener {

	private Plugin plugin;

	public Listeners(Plugin plugin) { // Notez le constructeur car sinon nous ne pourrions pas obtenir la config ! Nous ne pouvons obtenir la config uniquement avec un objet Plugin !
		this.plugin = plugin;
	}

	@EventHandler
	private void onPlayerJoin(PlayerJoinEvent event) {
		String message = plugin.getConfig().getString("messages.4").replace("/player/", event.getPlayer().getName());
		event.setJoinMessage(ChatColor.AQUA + message);
	}

}
```

Voilà voilà, prenez bien le temps de lire les commentaires et vous aurez tout compris ;)

Prenez également de tester tout les recoins de votre plugin, on sait jamais un bug peut toujours exister !

Voici également un tutoriel plus complet sur les configurations, mais je me répète, je vous conseille ma librairie ^^

VI – Planifier des tâches avec le Scheduler de Bukkit

Alors tout d'abord, que-ce-qu'un **Scheduler** ? D'après Google traduction, un **Scheduler** est un planificateur. Avec le **Scheduler** de Bukkit, nous pourrons donc planifier l'exécution de tâches. Il y a différentes types de tâches (se répétant, se planifiant, …). Pour voir tout ce qui est possible, je vous renvoi sur cette page (Vous avez vu ? C'est la **JavaDoc** de la version recommandée !).

Donc ici, nous allons simplement créer une tâche qui se répète et qui envoi « **Bonjour toi :3** » à un joueur au hasard !

Nous allons également apprendre à séparer la méthode **onCommand(...)** de la classe principale pour des raisons esthétiques. Et aussi enlever la variable plugin de la classe **Listeners**.

Donc alors par quoi allons nous commencer... Le **onCommand(...)** ? Très bien, alors créez une nouvelle classe « **Commands** » qui implémentera « **CommandExecutor** » et copiez tout votre bloc **onCommand(...)** dedans, on verra pour les erreurs après ;)

Ce qui nous donne :

```
public class Commands implements CommandExecutor {

	@Override
	public boolean onCommand(final CommandSender sender, Command cmd, String
label, String[] args) {
		Player player = null;

		if(sender instanceof Player) {
			player = (Player)sender;
		}
		else {
			return false;
		}

		if(cmd.getName().equalsIgnoreCase("hi")) {
			player.sendMessage(this.getConfig().getString("messages.2"));
			String message =
this.getConfig().getString("messages.3").replace("/player/", player.getName());
// Notez bien le replace(...), il n'y pas de fonction toute près faite pour
désigner un joueur !
			this.getServer().broadcastMessage(ChatColor.GREEN + message);
		}

		return true;
	}

}
```

Donc dans la classe « **HiMinecraftPlugin** », créez une variable **FileConfiguration** qui s'appelle « **config** ». Celle-ci devra être protégée (pour éviter les mauvaises surprises avec des plugins tiers). Elle devra aussi être statique pour pouvoir être utilisée dans les autres classes sans avoir besoin d'instancier la classe principale. Ce qui nous donne :

```
protected static FileConfiguration config;
```

Ensuite dans votre méthode **loadConfig()**, vous ajoutez :

```
config = this.getConfig(); // La valeur config est maintenant égale à la config
de notre plugin.
```

Et enlevez tout ce qui est **this.getConfig()** et remplacez par **config**, ça évite d'appeler la méthode **getConfig()** tout le temps, voici donc notre **loadConfig()** :

```
public void loadConfig() {
	config = this.getConfig();
	config.options().copyDefaults(true);
	this.saveConfig();
}
```

Ce qui nous permettra de charger notre configuration sans variable supplémentaire ! :D
Donc direction notre classe **Listeners** et enlevez la variable **plugin** ainsi que le constructeur de **Listeners**. Ensuite, changez le **plugin.getConfig()** par **HiMinecraftPlugin.config** ce qui permet de ne pas se trimballer la variable plugin pendant tout le long de l'exécution de la classe **Listeners**, on obtient ainsi ceci :

```
public class Listeners implements Listener {

	@EventHandler
	private void onPlayerJoin(PlayerJoinEvent event) {
		String message =
HiMinecraftPlugin.config.getString("messages.4").replace("/player/",
event.getPlayer().getName());
		event.setJoinMessage(ChatColor.AQUA + message);
	}

}
```

Attention, vous ne pouvez faire ceci uniquement sur les **variables statiques** ! (Mais vous devriez déjà le savoir si vous aviez suivi les cours sur le Java :P)

Vous obtiendrez une erreur lors de la création de la classe **Listeners**. Dans la classe principale, enlevez simplement l'argument **this**. Et remplacez aussi **this.getServer()** par **Bukkit** (ce qui nous donne une méthode statique), voici donc notre **onEnable()** :

```
@Override
public void onEnable() {
	loadConfig();
	Bukkit.getPluginManager().registerEvents(new Listeners(), this);
	System.out.println(config.getString("messages.1"));
}
```

Tant qu'on y est, vous avez remarqué un truc supplémentaire ? J'ai ajouté le **loadConfig()** dans notre **onEnable()** pour que la config soit chargée au démarrage ! (On avait oublié ça dans le chapitre précédent, j'espère que vous vous en êtes aperçu quand même ^^).

Voilà voilà, bon nous n'avons toujours pas corrigé les erreurs dans la classe **Commands**, donc allons regarder ça de plus près...

Tout d'abord je souhaite vous laisser réfléchir à la façon dont on s'y prendra pour corriger les erreurs, je tiens à préciser que nous venons de voir ça ;)

C'est bon ? Voici ma petite solution commentée :

```
public class Commands implements CommandExecutor {

	@Override
	public boolean onCommand(final CommandSender sender, Command cmd, String
label, String[] args) {
		Player player = null;

		if(sender instanceof Player) {
			player = (Player)sender;
		}
		else {
			return false;
		}

		if(cmd.getName().equalsIgnoreCase("hi")) {

	player.sendMessage(HiMinecraftPlugin.config.getString("messages.2")); //
Accès à la variable statique.
			String message =
HiMinecraftPlugin.config.getString("messages.3").replace("/player/",
player.getName()); // Idem ici aussi.
			Bukkit.broadcastMessage(ChatColor.GREEN + message); //
Maintenant que la classe n'étend plus la classe JavaPlugin, nous allons
utiliser méthode statique broadcastMessage(...) de la classe Bukkit.
		}

		return true;
	}

}
```

Voilà voilà, je ne détaillerai pas plus ici car il y a déjà pas mal de commentaires ^^
Vous pouvez tester, ça marche totalement !

Bon bon, passons maintenant à notre **Scheduler** : créez une nouvelle classe appelée « **HiTask** », elle implémentera « **Runnable** » (nous n'avons actuellement pas besoin de **BukkitRunnable**). Normalement, **Eclipse** vous demandera si vous souhaitez ajouter les méthodes manquantes, laissez le donc faire ;)

Maintenant, écrivons le code :
Il faut prendre un joueur au hasard, nous aurons donc besoin d'une variable **Random**. Nous allons donc créer la variable **rand**, il faut qu'elle soit privée ainsi que finale. Il faudra également instancier la classe **Random**.

Voici donc le code :

```
private final Random rand = new Random();
```

Dans notre run, nous exécutons la méthode qui nous permet de d'envoyer un message à un joueur au hasard.
Je vous laisse y réfléchir avant de regarder la suite... :P

Voici donc la méthode **run()** :

```
public class HiTask implements Runnable {

	private final Random rand = new Random();

	@Override
	public void run() {
		Player[] online = Bukkit.getOnlinePlayers(); // On obtient le
nombre de joueurs en ligne.
		if(online.length != 0) { // Si il y a des joueurs en ligne.
			online[rand.nextInt(online.length)].sendMessage("Bonjour
toi :3"); // On choisit le joueur au hasard et on lui envoi le message.
		}
	}

}
```

Si vous connaissez bien votre java, vous comprendrez comment on choisit un joueur au hasard mais si vous voulez je peux expliquer ;)
Donc dans la première ligne, on obtient le nombre de joueurs connectés et on le met dans un tableau de valeurs (à une dimension) d'objets **Player**.
Dans la deuxième on vérifie qu'il y a au moins un joueur en ligne.
Ensuite dans la troisième ligne, on choisit le joueur en fonction du **Random** (la valeur de **rand** est définie en fonction de la longueur du tableau de valeurs **online**). Ensuite on envoi le message !

Bon, c'est bien beau mais dans le chapitre précédent, on a apprit à faire une config non ?
Eh bien mettons le à profit !
Donc maintenant ajoutez un message à la config :

```
'5': Bonjour toi :3
```

Et chargez le dans le code :

```
online[rand.nextInt(online.length)].sendMessage(HiMinecraftPlugin.config.getStr
ing("messages.5")); // Toujours la variable statique.
```

Parfait parfait, il ne manque plus qu'a l'appeler dans notre **onEnable()** !
Nous souhaitons que la durée soit configurable donc nous n'avons qu'à ajouter la durée dans la config :

```
config:
  'hi-time': 600
```

Appelons le maintenant :P

```
Bukkit.getScheduler().scheduleSyncRepeatingTask(this, new HiTask(), 0,
config.getInt("config.hi-time") * 20L); // La durée est en tick, donc on
multiplie par 20 pour obtenir des secondes. Donc ici, 600 secondes.
```

Je vous laisse vous documenter sur cette fonction ici ;)
Voilà voilà ^^

Maintenant on test...

Oh ! La commande **/hi** ne marche plus ! Tout simplement car il faut définir l'exécuteur de la commande (vu qu'on l'a retiré de la classe principale) comme ça :

```
this.getCommand("hi").setExecutor(new Commands());
```

La classe **HiMinecraftPlugin** est maintenant comme ça :

```
public class HiMinecraftPlugin extends JavaPlugin {

	protected static FileConfiguration config;

	@Override
	public void onEnable() {
		loadConfig();
		Bukkit.getPluginManager().registerEvents(new Listeners(),
this);  // Laissez le this, on s'en servira après !
		System.out.println(config.getString("messages.1"));
		Bukkit.getScheduler().scheduleSyncRepeatingTask(this, new HiTask(),
0, config.getInt("config.hi-time") * 20L); // La durée est en tick, donc on
multiplie par 20 pour obtenir des secondes. Donc ici, 600 secondes.
		this.getCommand("hi").setExecutor(new Commands());
	}

	public void loadConfig() {
		config = this.getConfig();
		config.options().copyDefaults(true);
		this.saveConfig();
	}

}
```

Maintenant on peut re-tester...

Voilà voilà pour cette partie, pas mal notre petit plugin hein ? ^^
Je précise que nous pourrions par exemple ajouter des couleurs ou au lieu de « **Bonjour toi :3** » mettre « **Bonjour NomDuJoueur :3** » (mais vous savez désormais comment faire hein ? ;)).

VI – Trucs, astuces et optimisation du plugin

Tout d'abord, je tiens à m'excuser.
M'excuser pourquoi ? Car je ne vous ai pas tout expliqué.
Déjà, tout d'abord, pourquoi le mot-clé **static** sur la variable **config** ?
Tout simplement parce que on la « garde » tout au long du déroulement de notre plugin et qu'il peut être intéressant de l'avoir à tout moment sans avoir besoin d'instancier la classe qui la contient (dans notre cas **HiMinecraftPlugin**).

Voilà pour la variable statique, maintenant, pourquoi **protected** ?
Je vous renvoi sur ce tableau qui répond très bien aux question *Quand ? Comment ? Pourquoi ?* ;)

Ensuite, renommons le package **com.skyost.himinecraft** par **fr.skyost.himinecraft**. Pourquoi ? Car je ne possède pas le domaine « **www.skyost.com** » et que selon la convention de nomenclature Java, les domaines autorisés ne sont que : « **com, us, info, net et org** ». Je possède le domaine « **www.skyost.eu** » mais il n'est pas dans la liste ci-dessus. La solution ? Mettre les deux premières lettres de votre pays, en l'occurrence pour moi, « **fr** » (de plus, je possède le domaine « **www.skyost.fr** »).

/!\ Ne pas oublier de changer le nom du package dans le *main* du fichier *plugin.yml* /!

Maintenant, organisons un peu nos classes.
Renommez la classe **Commands** en « **CommandsExecutor** » (signifie : « Exécuteur de commandes » pour les anglophobes ;)).
Créez un package « **fr.skyost.himinecraft.listeners** » et glissez-y le fichier précédemment renommé. Vous devriez avoir une erreur.

Cherchez pourquoi, et si vous ne trouvez pas, je prépare ma boîte de Pringles...

Ça y est ? Trouvé ?
Oui, il faut changer le mot clé **protected** de la variable **config** en **public** car la classe **CommandsExecutor** ne se trouve plus dans la même classe que **HiMinecraftPlugin**.

Tiens, en parlant de **HiMinecraftPlugin**... Renommez la simplement en **HiMinecraft**, c'est juste plus joli ^^ (N'oubliez pas de modifier le **main** du plugin,yml !)

Renommez **Listeners** en « **EventsListener** » (« Écouteur d'événements ») et glissez le dans le dernier package créé.

Créez le package « **fr.skyost.himinecraft.tasks** » et glissez-y la classe « **HiTask** » puis ouvrez-la.
Attribuez le mot-clé **final** à la variable **online**, ce qui nous permet une meilleur rapidité dans l'exécution du code. (Pas des améliorations prodigieuses non plus hein !)

Ouvrez notre exécuteur de commandes puis attribuez le mot-clé **final** à tous les paramètres du **onCommand(...)** ce qui donne :

```
public boolean onCommand(final CommandSender sender, final Command cmd, final
String label, final String[] args)
```

Puis, enlevez la condition qui teste la valeur de la variable **cmd** car elle nous n'est d'aucune utilité. (en effet, notre plugin ne contient qu'une seule et unique commande, pourquoi tester la valeur de **cmd** alors ?)
Maintenant, copiez le contenue de la variable **message** dans la méthode **sendMessage(...)**.
Ce qui nous donne le code suivant :

```
@Override
public boolean onCommand(final CommandSender sender, final Command cmd, final
String label, final String[] args) {
	Player player = null;

	if(sender instanceof Player) {
		player = (Player)sender;
	}
	else {
		return false;
	}

	player.sendMessage(HiMinecraft.config.getString("messages.2")); // Accès
à la variable statique.
	Bukkit.broadcastMessage(ChatColor.GREEN +
HiMinecraft.config.getString("messages.3").replace("/player/",
player.getName())); // Maintenant que la classe n'étend plus la classe
JavaPlugin, nous allons utiliser la méthode statique broadcastMessage(...) de
la classe Bukkit.

	return true;
}
```

Sachez que ce code est plus efficace que l'ancien.

Ce chapitre ne contenait rien d'indispensable, uniquement des petits trucs utiles à savoir ;)
Pour le clore, je vous donne divers lien concernant les conventions de nomenclature en Java :

- Lien 1.
- Lien 2.
- Lien 3.

VII – Aller plus loin

Dans le cadre de ce plugin, vous pourriez par exemple ajouter une option dans la configuration permettant d'activer (ou non) les événements, voir juste activer le message de bienvenue, pour cela un simple **boolean** dans la config suffit ;)

Ce tutoriel n'était que sommaire, nous n'avons pas abordé toutes les notions de l'API et il nous reste encore beaucoup de travail.

Pour cela, je vous conseille fortement de chercher des plugins open source sur dev.bukkit.org (un moteur de recherche est disponible ici) mais je ne vous conseille pas de regarder les sources de plugins **hyper méga compliqué de la mort qui tue et qui déchire**. Non, regardez des sources simples et communes, comme des plugins « **Homes** » ou « **Warps** » (même système) par exemple ^^

A partir de maintenant, la limite à votre connaissance sera votre imagination car tout ce que vous ne connaissez pas, vous l'apprendrez.

Auteur : Skyost.

Suivez moi sur Twitter, Épinglez moi sur Pinterest, Regardez moi sur Youtube ou Retrouvez moi sur BukkitDev.

Merci à l'incroyable hébergeur de serveurs Minecraft : NiHost !

N'hésitez pas non plus à visiter mon site :D