Communication Inter-processus IPC et Multithreading

WORKSHOP

Ce workshop accompagne le prosit et vise à corriger étape par étape les problèmes de l'énoncé.

SETUP

La première phase de ce workshop est de constituer l'environnement de travail.

Vous allez avoir besoin :

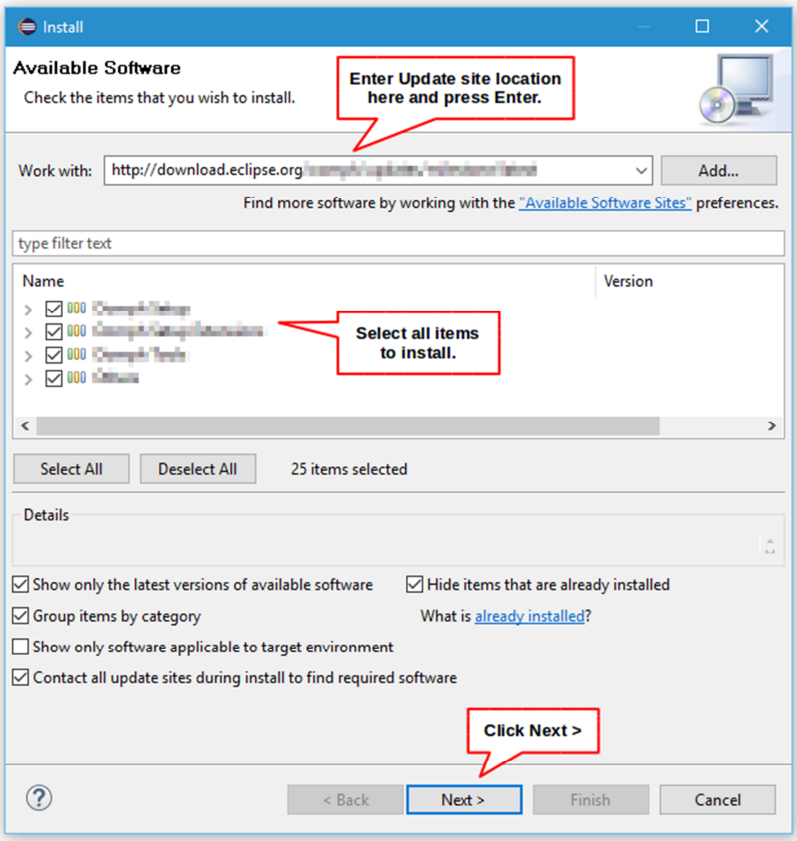
- D'un JDK Java 8 standard Edition (SE)
- D'un IDE type Eclipse ou Netbean. La suite de ce workshop se base sur Eclipse.

Une fois ces composants installés, lancer l'IDE et créer un nouveau projet.

INSTALLER WINDOWS BUILDER

Avant de commencer, il faut installer WindowsBuilder dans Eclipse, afin de pouvoir éditer graphiquement les IHM. Pour cela, aller dans le menu « Help > Install new Software », et entrer l'URL suivante :

http://download.eclipse.org/windowbuilder/WB/release/4.6/



Eclipse aura besoin de se relancer après l'installation.

Communication Inter-processus IPC et Multithreading

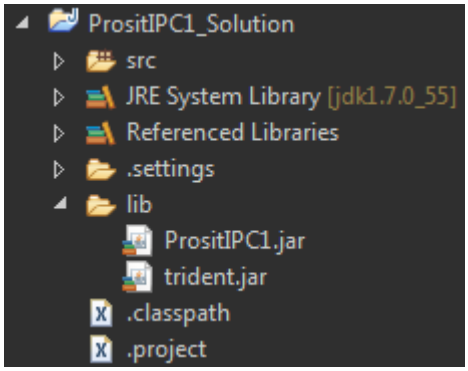
WORKSHOP

LANCER L'APPLICATION DE SIMULATION

Une fois le nouveau projet créé dans Eclipse, ajouter un répertoire lib dans le dossier de votre projet. Ce projet doit se trouver dans votre workspace (un répertoire utilisé par Eclipse pour enregistrer tous vos projets).

Dans ce répertoire lib, copier les deux librairies disponibles sur Moodle : PrositIPC1.jar et trident.jar

Appuyez sur F5 pour rafraichir l'explorateur de projet dans Eclipse. Votre projet doit ressembler à ça :



Sélectionner les deux fichiers JAR, et cliquer avec le bouton droit de la souris pour faire apparaitre le menu contextuel. Choisissez « *Build Path* > *Add to build path* » pour que les librairies soient chargées.

Créer un nouveau fichier avec un main, et lancez l'application simplement :

```
public class Main {
	public static void main(String[] args) {
		// On lance l'application
		PrositIPC.start();
	}
}
```

Si une fenêtre s'affiche, alors c'est bon vous pouvez commencer. Sinon, reprenez les étapes précentes.

Créez ensuite 3 classes qui correspondent aux trois exercices :

1. Une classe Etape1 qui implémente l'interface IStep1Strategy
2. Une classe Etape2 qui implémente l'interface IStep2Strategy
3. Une classe Etape3 qui implémente l'interface IStep3Strategy

Le compilateur va vous demander d'implémenter plusieurs méthodes. Ensuite dans le main, entrez le code suivant pour utiliser vos propres classes et solutionner les exercices :

```
public static void main(String[] args) {
	// On indique les classes des différents exercices
	PrositIPC.Step1 = new Etape1();
	// PrositIPC.Step2 = new Etape2();
	// PrositIPC.Step3 = new Etape3();
	// On lance l'application
	PrositIPC.start();
}
```

Au début, n'activez pas Step2 et Step3 → activez les au fur et à mesure de votre progression !

Communication Inter-processus IPC et Multithreading

WORKSHOP

ETAPE 1

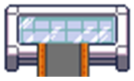


Figure 1 – Un quai de livraison

Figure 2 – Une machine X

La première étape, c'est de résoudre l'accès concurrent aux quais d'arrivée des matières premières. Pour cela, faites en sorte que les deux machines X ne puissent pas accéder en même temps aux quais, et si les quais sont vides.

La machine X transforme les produits M1 et M2 en une combinaison M3.

La lecture de cette ressource pourrait vous aider :
https://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html

Vous devez solutionner le problème en modifiant le code de la méthode onMachineRequest. Cette méthode est appelée en continue par les machines X quand elles souhaitent accéder aux matières premières venant des quais.

Actuellement, voici le code de cette méthode :

```
@Override
public Product onMachineRequest(InputDock dock, Machine machine) throws Exception
{

	return dock.accept();

}
```

Ce code pose des problèmes, les machines X lèvent de nombreuses erreurs comme vous pourrez le constater. A vous de voir comment vous pourriez améliorer les choses…

Une dernière consigne : vous n'avez pas le droit de faire des try/catch dans votre code, vous devez laisser passer les exceptions !

ETAPE 2

Figure 3 – La machine Y

La seconde étape vise à résoudre le goulot d'étranglement vers la machine Y. Si vous ne faites pas attention, les machines X envoient leurs travaux à la machine Y sans vérification, ce qui provoque des erreurs.

La machine Y transforme les produits M3 en produit M4.

Voici le code actuellement utilisé :

```
@Override
public void onMachineRequest(MachineX applicant, MachineY executor)
        throws Exception
{
}

@Override
public void onMachineExecute(MachineX applicant, MachineY executor)
        throws Exception
{
       executor.executeJob();
}
```

La méthode onMachineRequest est appelée quand la machine X a terminé, et s'apprête à acheminer sa marchandise vers la machine Y. Elle est sensée bloquer jusqu'à ce que la machine Y soit disponible. Actuellement, il n'y a aucun contrôle et certains colis sont envoyés vers la machine Y alors qu'elle n'est pas encore disponible.

La méthode onMachineExecute sert à lancer le travail de la machine une fois la marchandise arrivée à la machine. Là aussi, le lancement est fait sans aucune vérification…

Peut-être que la lecture de cet documentation pourra vous aider :
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html

ETAPE 3

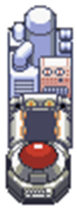


Figure 4 – Une série de machine Z

La dernière étape va améliorer le fonctionnement des machines Z. Les machines Z transforment les produits M4 en produits M5.

Actuellement, voici le code utilisé :

```
public MachineZ chooseMachine(MachineZ target1, MachineZ target2) throws Exception
{
	return target1;
}

public void onMachineRequest(Product product, MachineZ m1, MachineZ m2,
	MachineZ m3) throws Exception
{
	m1.executeJob(product);
	m2.executeJob(product);
	m3.executeJob(product);
	product.makeFinished();
}
```

La première méthode permet de choisir la série de machine Z qui sera utilisée. Vous devez renvoyer la série choisie. La seconde méthode sert à organiser les 3 machines Z au sein de la série choisie.

On constate bien que toutes les opérations sont faites les unes à la suite des autres. On aimerait bien paralléliser ces tâches, mais elles ne prennent pas toutes le même temps.

La première fait 300 millisecondes, la seconde fait 2100 ms, et la dernière 1500 ms.

Vous allez devoir vous creuser les méninges cette fois…

Si vous vous débrouillez bien, vous devriez atteindre plus de 150 pièces d'or par 30 secondes !