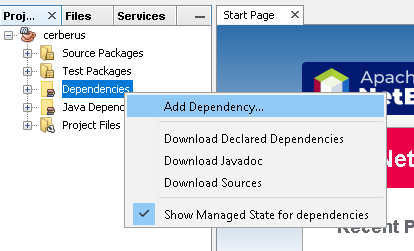
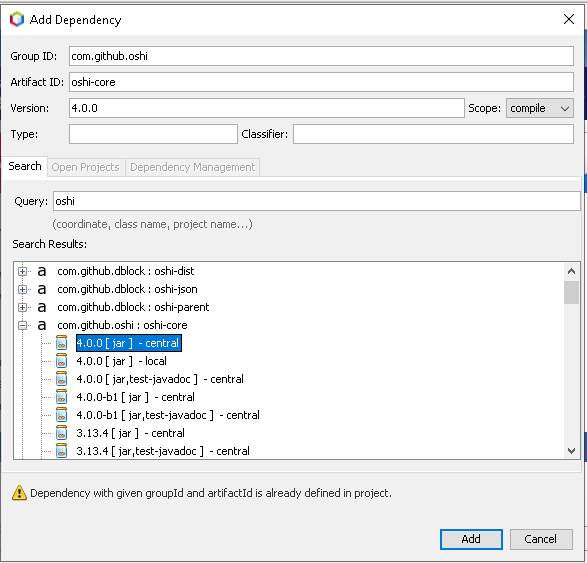
**UTILIZANDO A BIBLIOTECA OSHI**

**Primeiro passo**: Criar projeto com Maven no netbeans e instalar as dependências:

* Crie um novo projeto Java + Maven (semelhante aos projetos criados na aula do professor Yoshi).
* No canto direito da IDE, onde todas as pastas do projeto estão sendo mostradas, clique em “Dependências” com o botão direito do mouse e siga para “Adicionar Dependência”



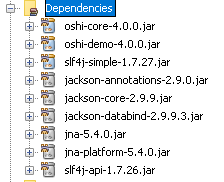
* Na janela que se abrirá você poderá ver um campo com o nome “Query”, nele, você procura as bibliotecas necessárias ao seu projeto. Nesse exemplo, as bibliotecas que instalaremos serão a **oshi-demo** e a **oshi-core** (Para isso pesquise o nome da biblioteca, a localize nos resultados, pegue sua versão mais recente e clique em “Adicionar”).



Versões da biblioteca

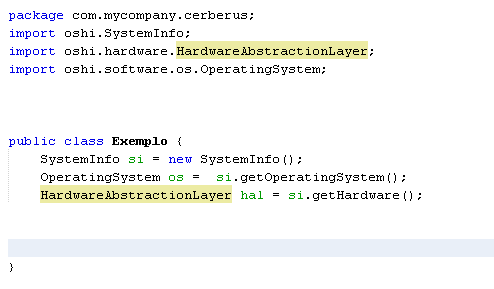
Campo de pesquisa

* Com as dependências instaladas, você pode ver que mais de duas bibliotecas foram adicionadas ao projeto.

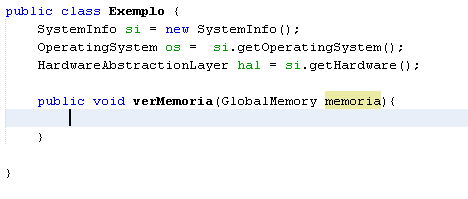


**Segundo Passo**: Criar uma classe e utilizar a biblioteca e coletar dados da memória.

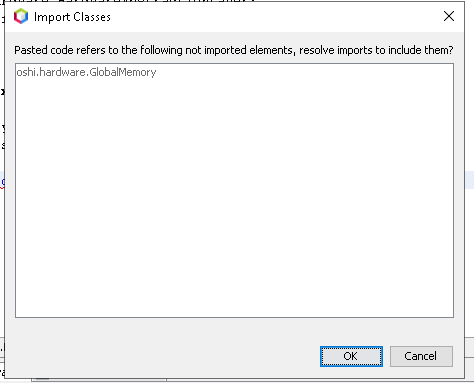
* Crie uma classe java, no exemplo criado eu dei ao arquivo o nome de “Exemplo”. Para coletar qualquer tipo de dados que a oshi permite, precisamos instanciar duas classes, a primeira **SystemInfo** (É dela que vão “sair” as outras duas intarfaces importantes: **OperatingSystem** e **HardwareAbstractionLayer**, a primeira, nos fornece informações relacionadas ao sistema operacional, já a segunda, informações a respeito do hardware do computador analisado). A cada instancia feita, a IDE solicitará a importação dessas classes no arquivo, no final, o arquivo ficará parecido com isso:



* Agora já podemos criar métodos para coletar as informações da memória, nesse exemplo, vou criar um método chamado “verMemoria()” que ganhará como parâmetro um objeto Memória da interface HardwareAbstractionLayer que já foi instanciada (dessa forma o código fica um pouco mais organizado):



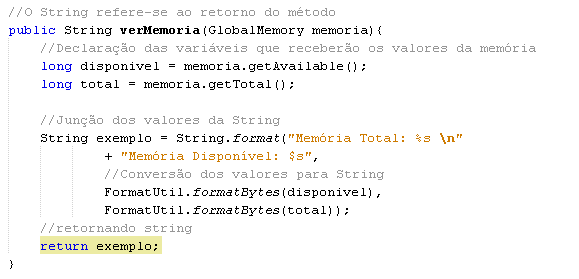
**obs:** Ao escrever o objeto ‘Global Memory’, a própria IDE já pede que essa classe seja importada ao código: (se ela não pediu, importe manualmente)



* Agora, dentro do nosso método, criarei variáveis do tipo “long” que receberão todos os dados da memória, no exemplo, vou coletar a memória total e a disponível. A utilização do tipo “long” ocorre pois esse é o tipo de retorno dos métodos chamados.

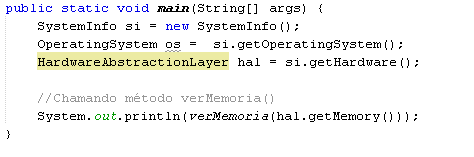


* Agora, vamos juntar essas variáveis em uma única String que retornará quando chamarmos esse método, para isso preciamos:
* Mudar o retorno do método de “void” para String
* Converter os valores vindos da memória, pois eles chegam em formato de bytes (Para isso, utilizarei uma classe da oshi chamada **FormatUtil**, que deve ser importada). O código ficará dessa maneira:

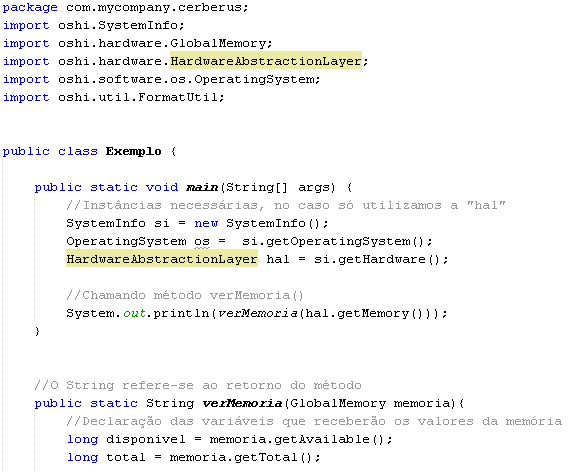


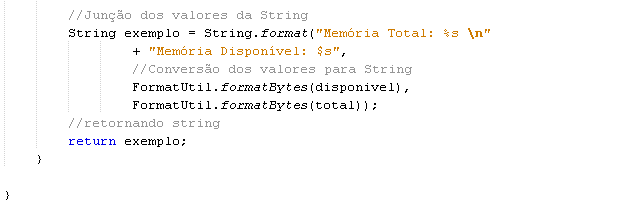
**Terceiro Passo:** Criarclasse main e executar o método.

* Dentro do mesmo arquivo, vou criar uma classe main e apenas imprimir no console o retorno do método **verMemoria,** não esquecendo, que o método precisa de um parâmetro do tipo **GlobalMemory,** que é pego através da classe **HardwareAbstractionLayer:**



No final, o código fonte ficou da seguinte maneira:





Ao ser executado, esse é o resultado:



**Dicas:** Para descobrir outras funcionalidades dos objetos, com o Ctrl pressionado, clique no nome das classes instanciadas, assim você poderá ver todos os métodos disponíveis para serem chamados. No github da oshi, você acha detalhamento de como os métodos respondem e de como formatar as informações recebidas, além de outros exemplos de utilização da biblioteca.

Exemplos de utilização:

<https://github.com/oshi/oshi/blob/master/oshi-core/src/test/java/oshi/SystemInfoTest.java>

Git da oshi (home):

<https://github.com/oshi/oshi>

Pastas com classes: (hardware, software(SO) e util(formatações) )

<https://github.com/oshi/oshi/tree/master/oshi-core/src/test/java/oshi>