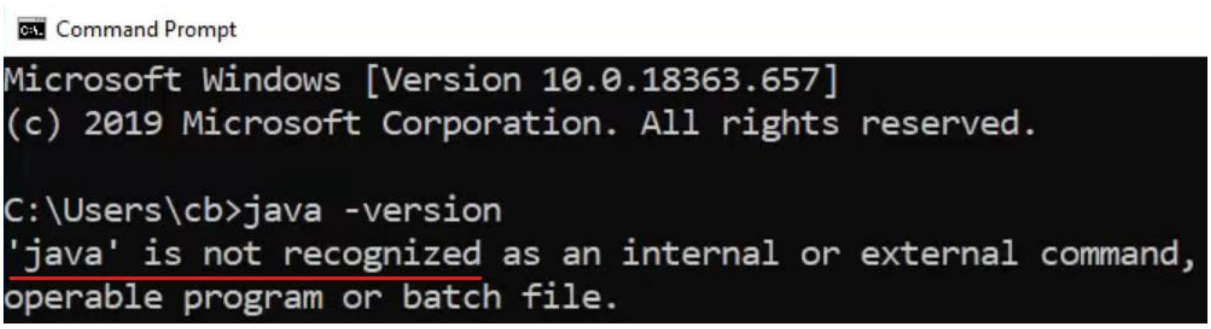
COS30018 – Intelligent Systems

The basics of Choco

1. Prerequisites:

The requirements to run Choco are to have **JDK 8+** and **Maven 3+** installed on your machine.

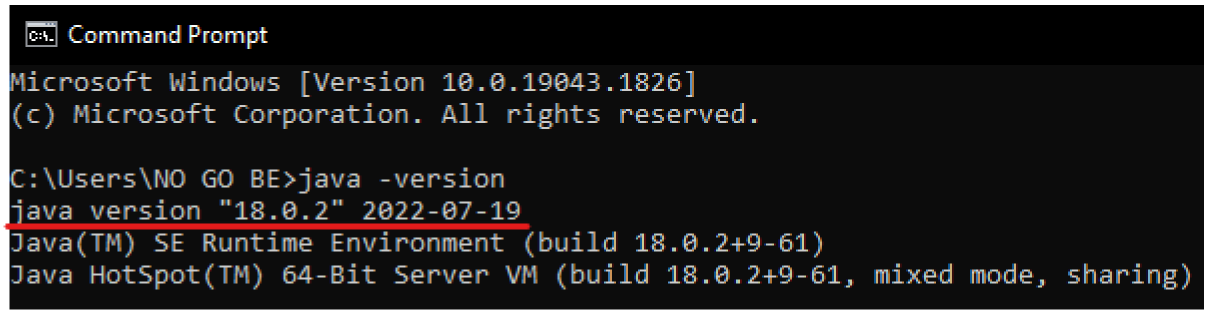
To check if you have the JDK 8+ installed, open your command line/terminal and type "java -version". If "java" is not recognised, it means you haven't got JDK installed. Please follow the instruction "How to install JDK" in Canvas under the Tutorial Resources tab to download and install JDK on your machine.



```
Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\cb>java -version
'java' is not recognized as an internal or external command,
operable program or batch file.
```

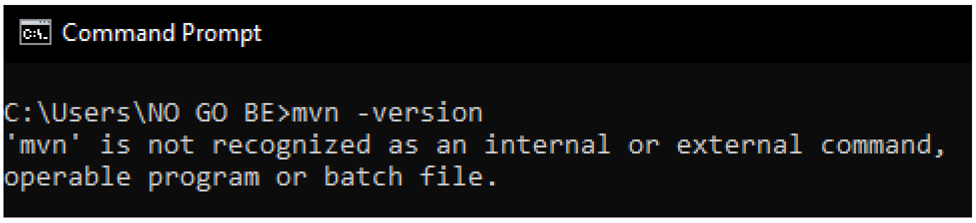
If the version is lower than 8, please also follow the same instruction to download the latest version of JDK.

If the version is greater than 8, then you are good to go.



```
Command Prompt
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\NO GO BE>java -version
java version "18.0.2" 2022-07-19
Java(TM) SE Runtime Environment (build 18.0.2+9-61)
Java HotSpot(TM) 64-Bit Server VM (build 18.0.2+9-61, mixed mode, sharing)
```

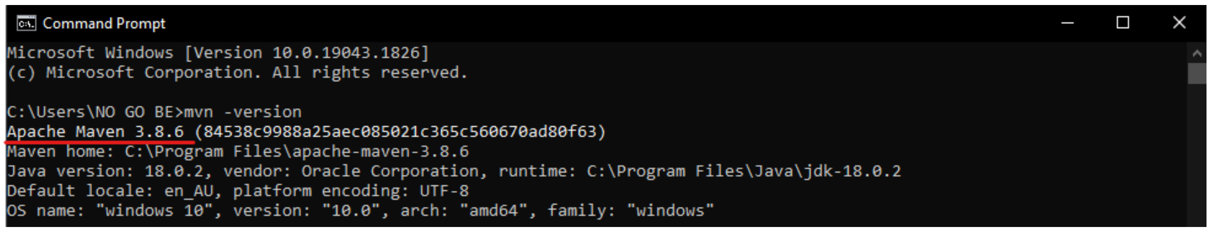
To check if you have the Maven 3+ installed, open your command line/terminal and type "mvn -version". If "mvn" is not recognised, it means you haven't got Maven installed. Please follow the instruction "How to install Maven" in Canvas under the Tutorial Resources tab to download and install Maven on your machine.



```
Command Prompt

C:\Users\NO GO BE>mvn -version
'mvn' is not recognized as an internal or external command,
operable program or batch file.
```

If the version is lower than 3, please also follow the same instruction to download the latest version of Maven.

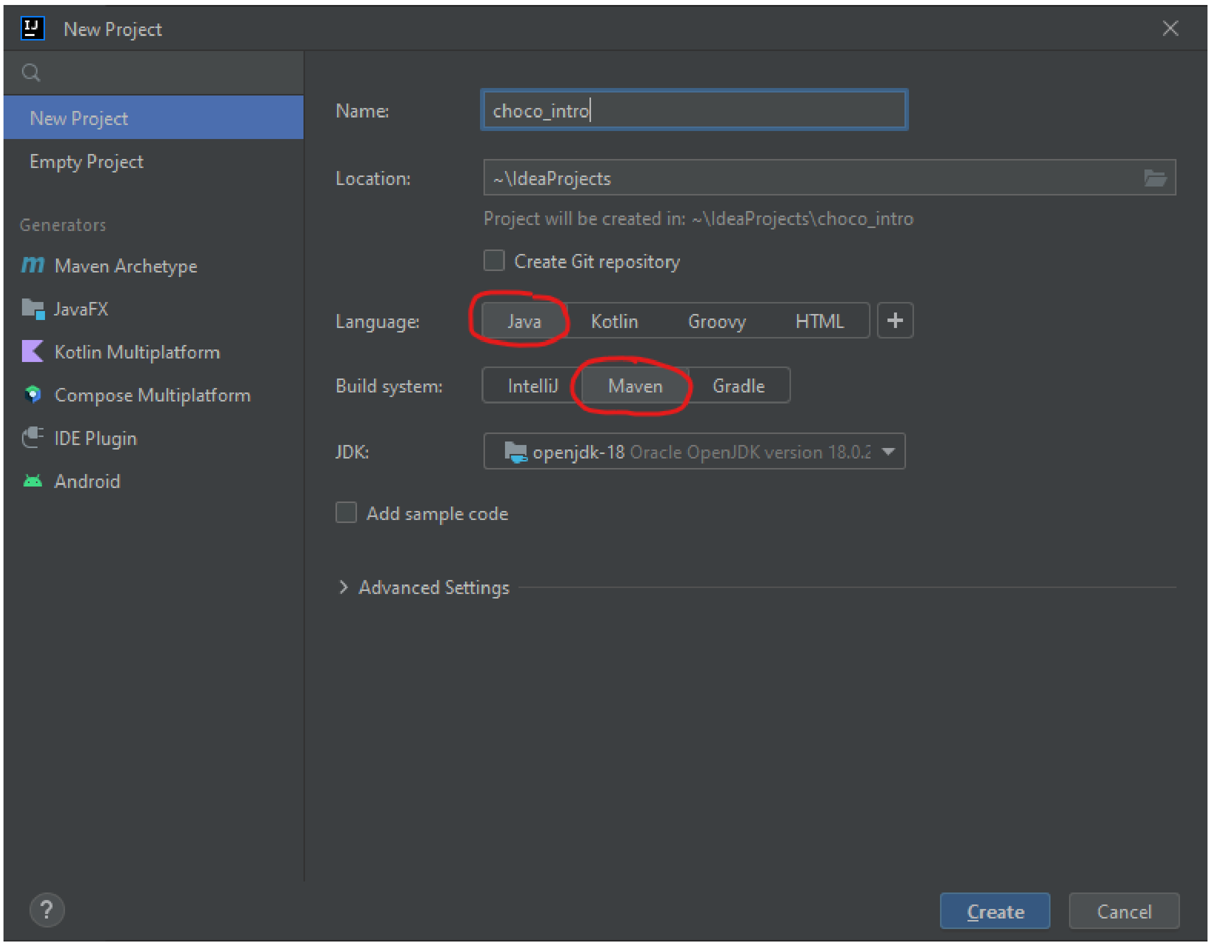
If the version is greater than 3, then you are good to go.



```
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\NO GO BE>mvn -version
Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
Maven home: C:\Program Files\apache-maven-3.8.6
Java version: 18.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-18.0.2
Default locale: en_AU, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

2. How to start

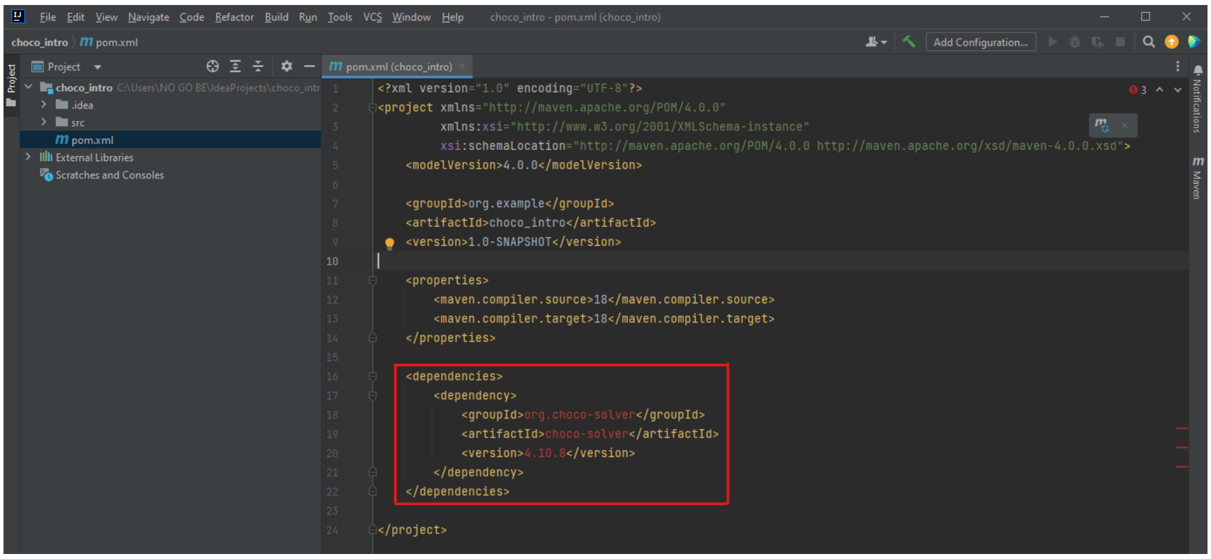
Open your IntelliJ IDE and create a new Java project with Maven:



Name: choco_intro
Location: ~\IdeaProjects
Project will be created in: ~\IdeaProjects\choco_intro
Create Git repository [ ]
Language: [✓ Java] Kotlin Groovy HTML
Build system: IntelliJ [✓ Maven] Gradle
JDK: openjdk-18 Oracle OpenJDK version 18.0.2
Add sample code [ ]

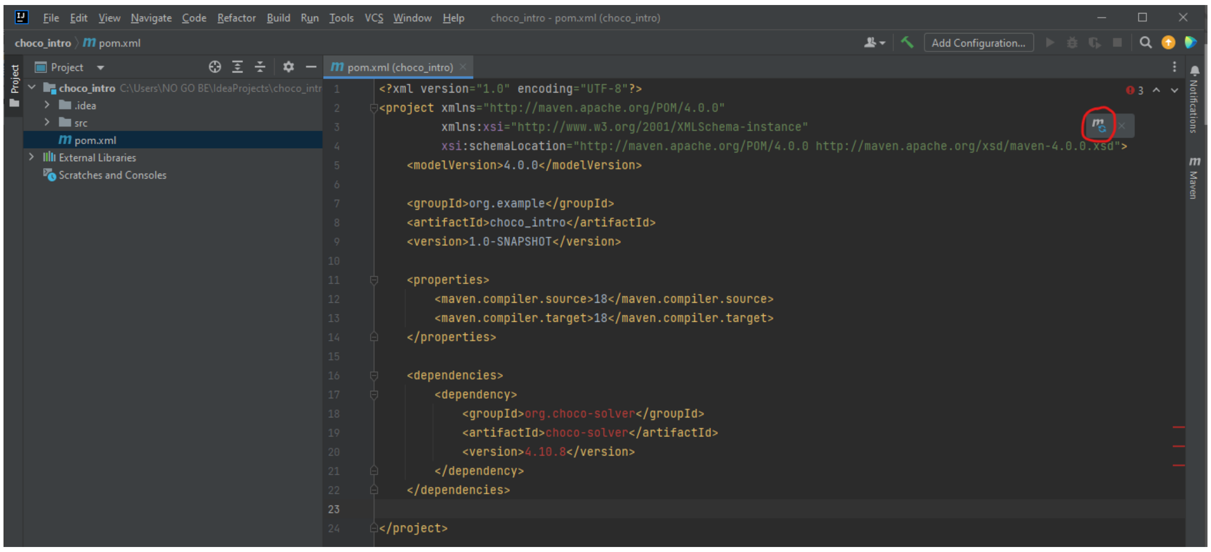
In the file hierarchy on the left, you will see a "pom.xml" file, create a <dependencies> tag and paste the following code inside that tag to download the Choco library for your project as shown below:

```xml
<dependency>
   <groupId>org.choco-solver</groupId>
   <artifactId>choco-solver</artifactId>
   <version>4.10.8</version>
</dependency>
```


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
choco_intro - pom.xml (choco_intro)
choco_intro › pom.xml
Add Configuration...
Project
choco_intro C:\Users\NO GO BE\IdeaProjects\choco_intr
.idea
src
pom.xml
External Libraries
Scratches and Consoles
pom.xml (choco_intro)
Notifications
Maven

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>choco_intro</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>18</maven.compiler.source>
        <maven.compiler.target>18</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.choco-solver</groupId>
            <artifactId>choco-solver</artifactId>
            <version>4.10.8</version>
        </dependency>
    </dependencies>

</project>
```

Save the file and click on the “Load Maven Changes” icon to start downloading the Choco library:


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
choco_intro - pom.xml (choco_intro)
choco_intro › pom.xml
Add Configuration...
Project
choco_intro C:\Users\NO GO BE\IdeaProjects\choco_intr
.idea
src
pom.xml
External Libraries
Scratches and Consoles
pom.xml (choco_intro)
Notifications
Maven

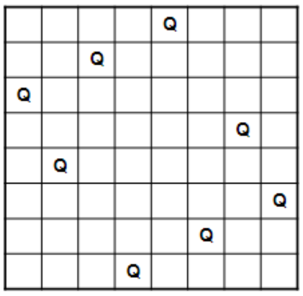
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>choco_intro</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>18</maven.compiler.source>
        <maven.compiler.target>18</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.choco-solver</groupId>
            <artifactId>choco-solver</artifactId>
            <version>4.10.8</version>
        </dependency>
    </dependencies>

</project>
```

Now Choco is loaded and you can start implementing your CSP now.

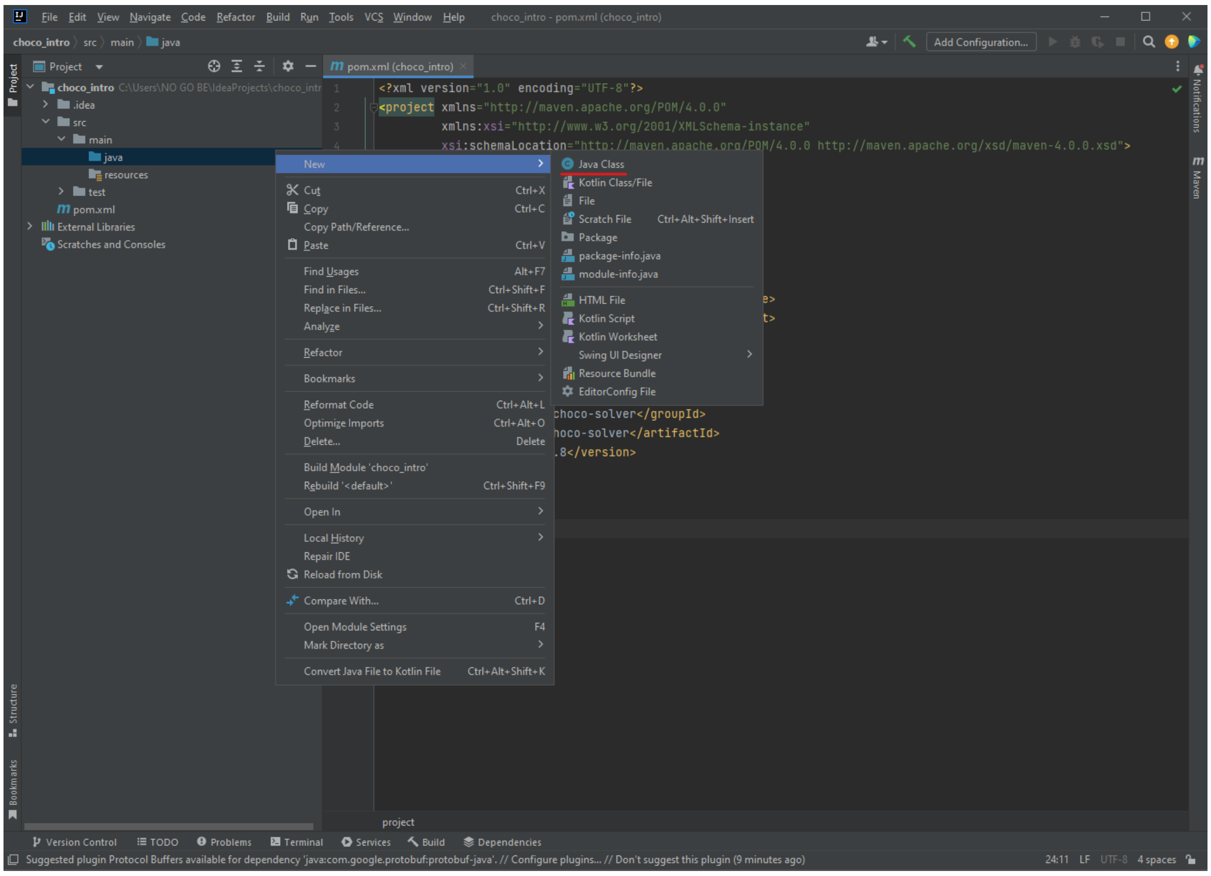
3. Choco for n-queens problem:

The n-queens problem is the problem of finding a way to place n queens on a $n \times n$ chessboard so that none attack each other.

An example solution for a 8×8 chessboard is as below:

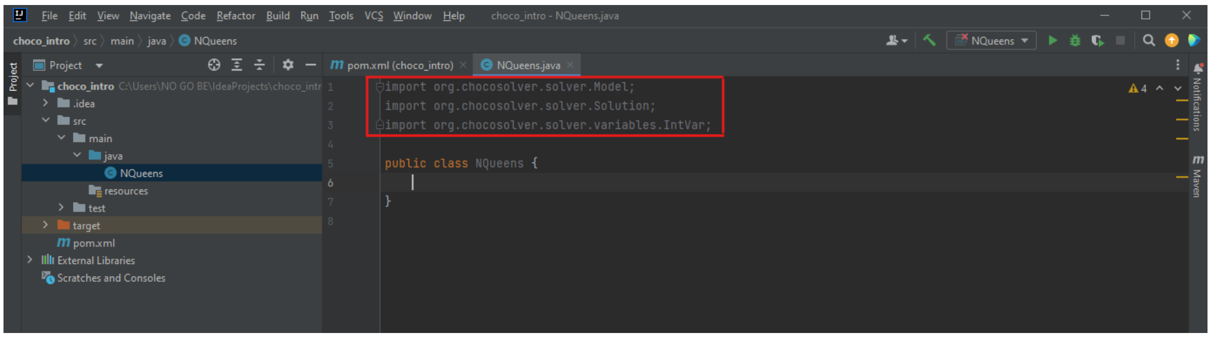


To solve this problem using Choco, firstly create a Java class in IntelliJ and name it NQueens:



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

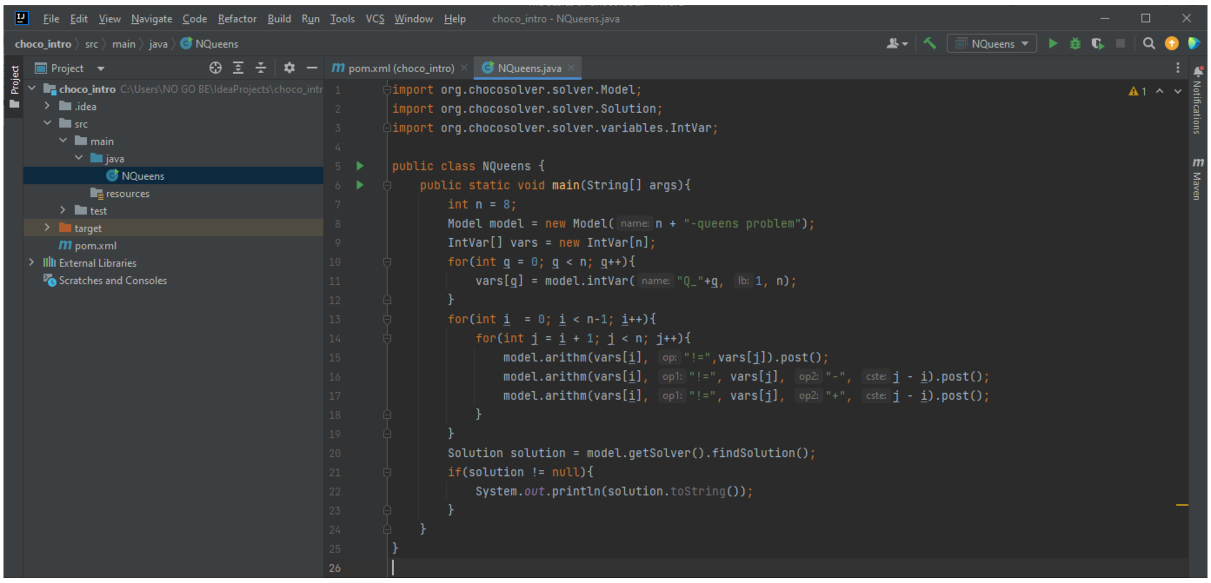
Then you need to import the Model, Solution, and variables.IntVar from Choco to demonstrate this example by using “import”:



```java
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.variables.IntVar;

public class NQueens {

}
```

Create a main function and paste the code given below to start it:

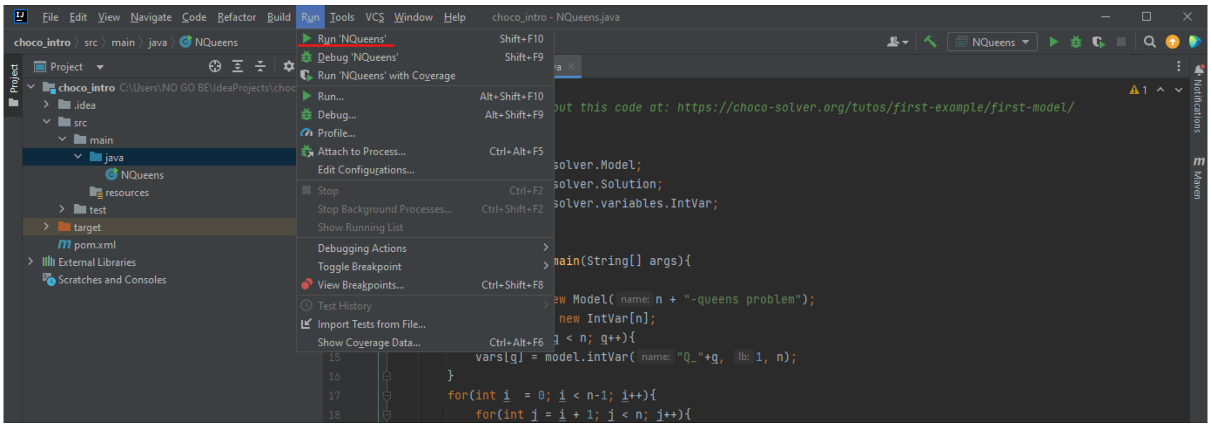
```
int n = 8;
Model model = new Model(n + "-queens problem");
IntVar[] vars = new IntVar[n];
for(int q = 0; q < n; q++){
    vars[q] = model.intVar("Q_"+q, 1, n);
}
for(int i  = 0; i < n-1; i++){
    for(int j = i + 1; j < n; j++){
        model.arithm(vars[i], "!=",vars[j]).post();
        model.arithm(vars[i], "!=", vars[j], "-", j - i).post();
        model.arithm(vars[i], "!=", vars[j], "+", j - i).post();
    }
}
Solution solution = model.getSolver().findSolution();
if(solution != null){
    System.out.println(solution.toString());
}
```



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
choco_intro - NQueens.java
choco_intro 〉 src 〉 main 〉 java 〉 NQueens
Project
choco_intro C:\Users\NO GO BE\IdeaProjects\choco_intr
.idea
src
main
java
NQueens
resources
test
target
pom.xml
External Libraries
Scratches and Consoles
pom.xml (choco_intro)
NQueens.java

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.variables.IntVar;

public class NQueens {
    public static void main(String[] args){
        int n = 8;
        Model model = new Model( name: n + "-queens problem");
        IntVar[] vars = new IntVar[n];
        for(int q = 0; q < n; q++){
            vars[q] = model.intVar( name: "Q_"+q,  lb: 1, n);
        }
        for(int i  = 0; i < n-1; i++){
            for(int j = i + 1; j < n; j++){
                model.arithm(vars[i],  op: "!=",vars[j]).post();
                model.arithm(vars[i],  op1: "!=", vars[j],  op2: "-",  cste: j - i).post();
                model.arithm(vars[i],  op1: "!=", vars[j],  op2: "+",  cste: j - i).post();
            }
        }
        Solution solution = model.getSolver().findSolution();
        if(solution != null){
            System.out.println(solution.toString());
        }
    }
}
```

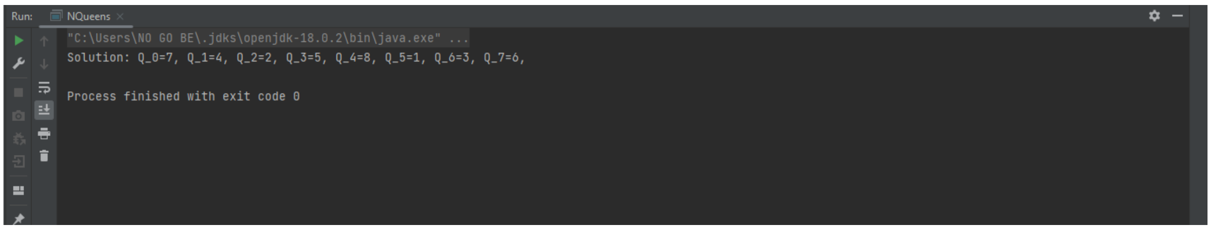
Let’s run the program to see the output first, then we will explain each component later. Click on Run “NQueens” or Shift + F10 to run it:



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
choco_intro - NQueens.java
Run 'NQueens' Shift+F10
Debug 'NQueens' Shift+F9
Run 'NQueens' with Coverage
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Profile...
Attach to Process... Ctrl+Alt+F5
Edit Configurations...
Stop Ctrl+F2
Stop Background Processes... Ctrl+Shift+F2
Show Running List
Debugging Actions
Toggle Breakpoint
View Breakpoints... Ctrl+Shift+F8
Test History
Import Tests from File...
Show Coverage Data... Ctrl+Alt+F6

```
ut this code at: https://choco-solver.org/tutos/first-example/first-model/
solver.Model;
solver.Solution;
solver.variables.IntVar;
main(String[] args){
w Model( name: n + "-queens problem");
new IntVar[n];
q < n; q++){
    vars[q] = model.intVar( name: "Q_"+q,  lb: 1, n);
}
for(int i  = 0; i < n-1; i++){
    for(int j = i + 1; j < n; j++){
```

You will see a solution displayed:



```
"C:\Users\NO GO BE\.jdks\openjdk-18.0.2\bin\java.exe" ...
Solution: Q_0=7, Q_1=4, Q_2=2, Q_3=5, Q_4=8, Q_5=1, Q_6=3, Q_7=6,

Process finished with exit code 0
```

- The queen on the 0th row is placed on the 7th column
- The queen on the 1st row is placed on the 4th column
- …

Explanations:

The Model:

On line 8, a `Model` instance is declared. It is the key component of the library and needed to describe any problem. The syntax to declare a `Model` in Choco is:

```
Model model = new Model("my problem");
```

Once the model is created, variables and constraints can be defined.

The variables:

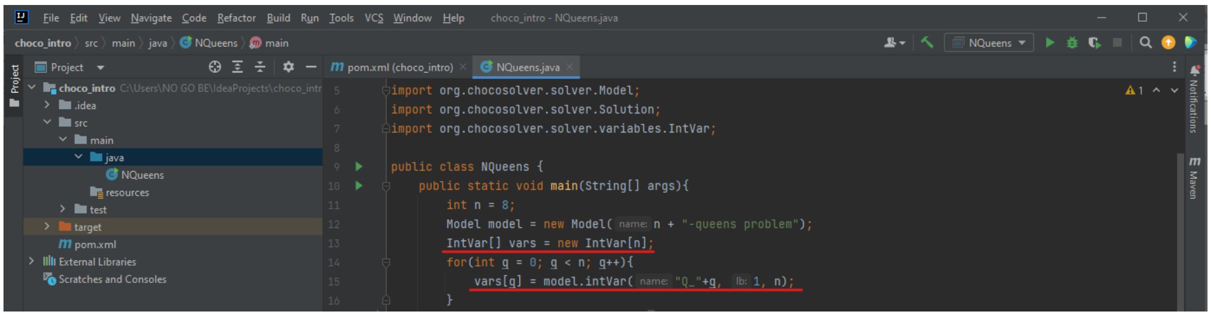
In CSP configuration, each problem is described with a set of variables and our goal is to *assign* a value to each variable. The variables **MUST** be created in a certain Model object. When creating a variable, the user can specify a name to help reading the output.

Choco-solver includes several types of variables: BoolVar, IntVar, SetVar and RealVar, but you will probably only need to use either IntVar or BoolVar since we just deal with discrete variables CSPs in this unit. You can read more about variables in Choco here: https://choco-solver.org/docs/modeling/variables/.

For the example above, a queen position is defined by its coordinates on the chessboard. Naturally, we don't know yet where to put queens on the chessboard, but we can give indications. To do so, we need to declare variables.

Here, in the solution, there will be exactly one queen per row (and per column). So, a modelling trick is to fix the row a queen can go to and only question on their column. Thus, there will be n queens (one per row), each of them to be assigned to one column, among $[1, n]$.

Lines 9 and 11 in the code managed to create variables and their domain.



```
5   import org.chocosolver.solver.Model;
6   import org.chocosolver.solver.Solution;
7   import org.chocosolver.solver.variables.IntVar;
8
9   public class NQueens {
10      public static void main(String[] args){
11          int n = 8;
12          Model model = new Model( name: n + "-queens problem");
13          IntVar[] vars = new IntVar[n];
14          for(int q = 0; q < n; q++){
15              vars[q] = model.intVar( name: "Q_"+q,  lb: 1, n);
16          }
```

The constraints:

The queens' position must follow some rules. We already encoded that there can only be one queen per row. Now, we have to ensure that, on any solution, no two queens share the same column and diagonal.

First, the columns conditions: if the queen i is on column k, then any other queens cannot take the value k. So, for each pair of queens, the two related variables cannot be assigned to the same value. This is expressed by the constraint on line 19. To activate the constraint, it has to be **posted**.

Second, the diagonals: we have to consider the two orthogonal diagonals. If the queen i is on column k, then, the queen $i + 1$ cannot be assigned to $k + 1$. More generally, the queen $i + m$ cannot be assigned to $k + m$. The same goes with the other diagonal. This is declared on line 20 and 21.

Choco has several types of constraints, but you will probably find the *arithm* and *allDifferent* constraints are enough for most problems:

- *allDifferent*: ensures that all variables take a distinct value in a solution.
- *arithm:* to express arithmetical relationships between two or three variables, or between one or two variables and an integer constant.

See more about Choco constraints here: https://javadoc.io/static/org.choco-solver/choco-solver/4.10.3/org.chocosolver.solver/org/chocosolver/solver/constraints/IConstraintFactory.html

Solving the problem:

Once the problem has been described into a model using variables and constraints, its satisfaction can be evaluated, by trying to solve it.

This is achieved on line 24 by calling the `getSolver().findSolution()` method from the model. If a solution exists, it is printed on the console.

Some improvements:

Variables:

First, lines 13-16 can be compacted into:

```
IntVar[] vars = model.intVarArray("Q", n, 1, n, false);
```

Doing so, an n-array of variables with $[1, n]$-domain is created. Each variable name is “$Q[i]$” where i is its position in the array, starting from 0. The last parameter, set to false, indicates that the domains must be enumerated (not bounded).

Constraints:

Lines 19 to 21 can be replaced by:

```
vars[i].ne(vars[j]).post();
vars[i].ne(vars[j].sub(j - i)).post();
vars[i].ne(vars[j].add(j - i)).post();
```

Where *ne* stands for *not equal*. Theses instructions express the same constraints, or more complex expressions, in a convenient way.

Or differently, if we view the problem in another perspective:

- There must be only one queen on each row (already encoded when we declare variables)
- There must be only one queen on each column
- There must be only one queen on each diagonal

By that, we can rewrite the constraints to:

```
IntVar[] diag1 = new IntVar[n];
IntVar[] diag2 = new IntVar[n];
for(int i = 0 ; i < n; i++){
    diag1[i] = vars[i].sub(i).intVar();
    diag2[i] = vars[i].add(i).intVar();
}
model.post(
    model.allDifferent(vars),
    model.allDifferent(diag1),
    model.allDifferent(diag2)
);
```

The first constraint simply states that all variables from `vars` must be different. The next 2 constraints state that all variables from a diagonal must be different. The variables of a diagonal are given by expressions of `diag1` and `diag2`.

4. Other practice:

After running and understanding the example above, you can now work on the Traveling Salesman Problem tutorial here: https://choco-solver.org/tutos/traveling-salesman-problem/. This will be very useful for those of you doing the VRP assignment.