

Programmation Orientée Objet (OBJET)

TP 1 : Présentation et prise en main de NetBeans (création d'un projet, écriture d'un premier programme et débogage)



Jean-Marie Normand – Bureau A114
jean-marie.normand@ec-nantes.fr


Instructions

- Suivez les slides les uns après les autres
- A la fin de cette séance de TP, **vous devrez nous rendre un rapport par binôme**
- Ce rapport devra contenir :
 - Une introduction et une présentation rapide du sujet de la séance
 - Les réponses aux questions posées dans les slides repérés par une icône de panneau STOP
 - Une conclusion
- La notation tiendra compte du respect de ces consignes



1^{RE} PARTIE : CRÉATION D'UN PROJET JAVA

Introduction

- Création d'un projet avec l'environnement de développement intégré (ou « Integrated Development Environment », IDE en anglais) NetBeans : <https://netbeans.org>  **NetBeansIDE**
- Si vous le souhaitez, vous pouvez utiliser un autre IDE (Eclipse, IntelliJ, etc.) mais sachez que l'équipe pédagogique ne fournira pas de documents présentant la création et la prise en main de ces autres outils.

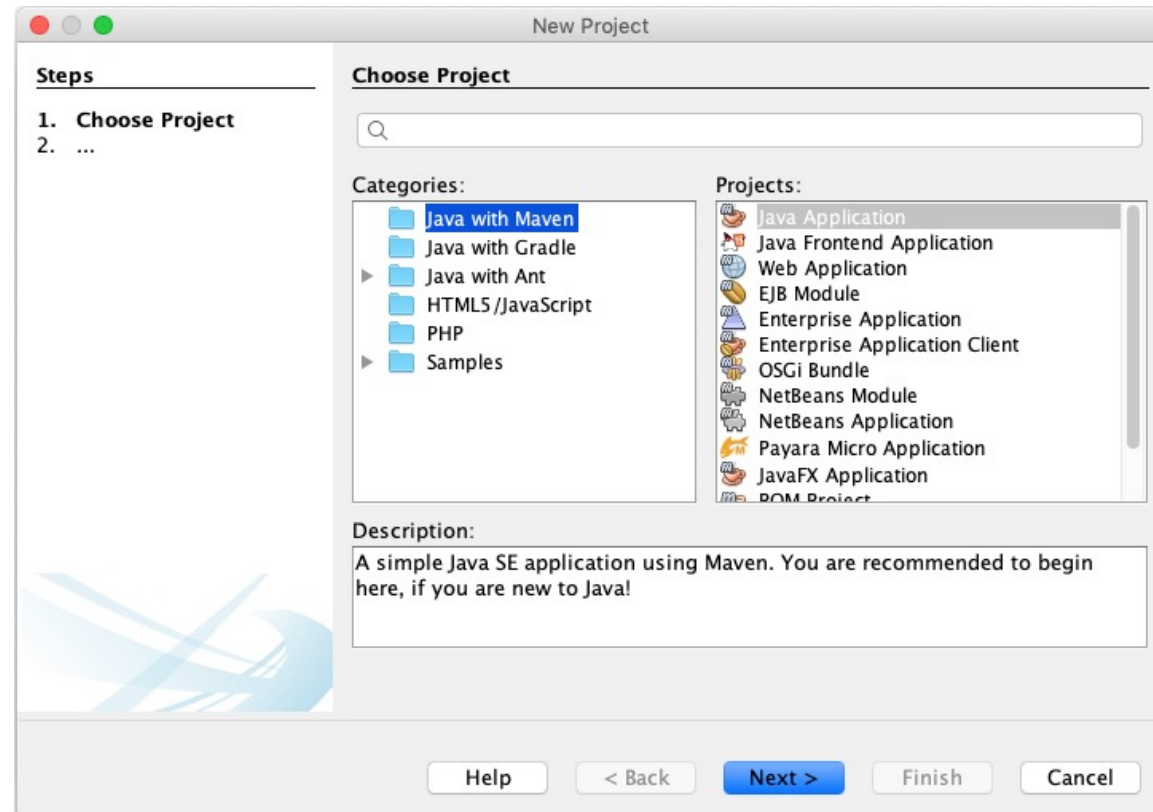
Création d'un projet

1. Lancer **NetBeans**

- Cliquer sur l'icône sur votre bureau ou allez chercher le raccourci dans vos menus

2. Créer un nouveau projet Java

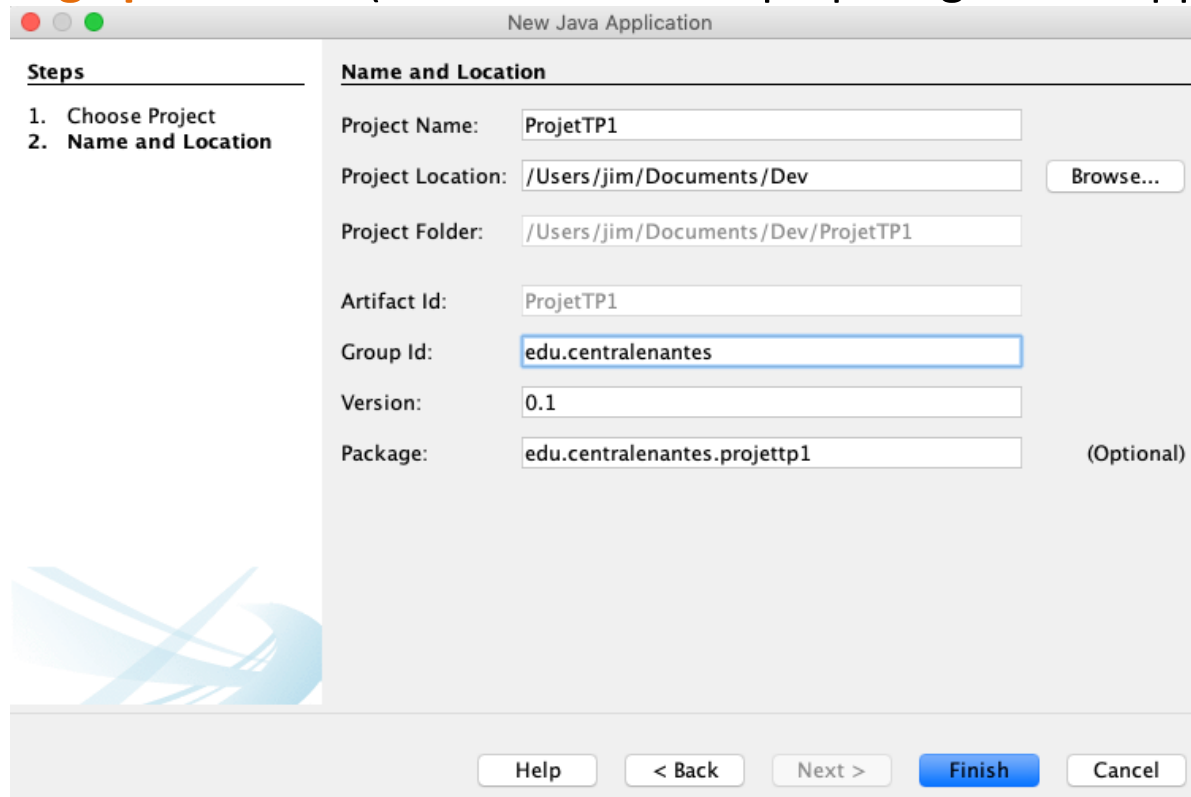
- Aller dans **File → New Project** (ou **Fichier → Nouveau Projet** si votre interface est en français)
- Dans la fenêtre pop-up choisissez **Java** puis **Java Application**



Création d'un projet (2)

3. Choisir le nom et les informations relatives à votre projet

- **Nom** de projet,
- **Endroit** où les fichiers sont stockés,
- Identifiant du Groupe du Projet
- **Version** du projet (obligatoire)
- **Nom du paquetage par défaut** (voir slides sur les paquetages sur Hippocampus)



The screenshot shows the 'New Java Application' dialog box. On the left, the 'Steps' pane lists '1. Choose Project' and '2. Name and Location'. The main area is titled 'Name and Location' and contains the following fields:

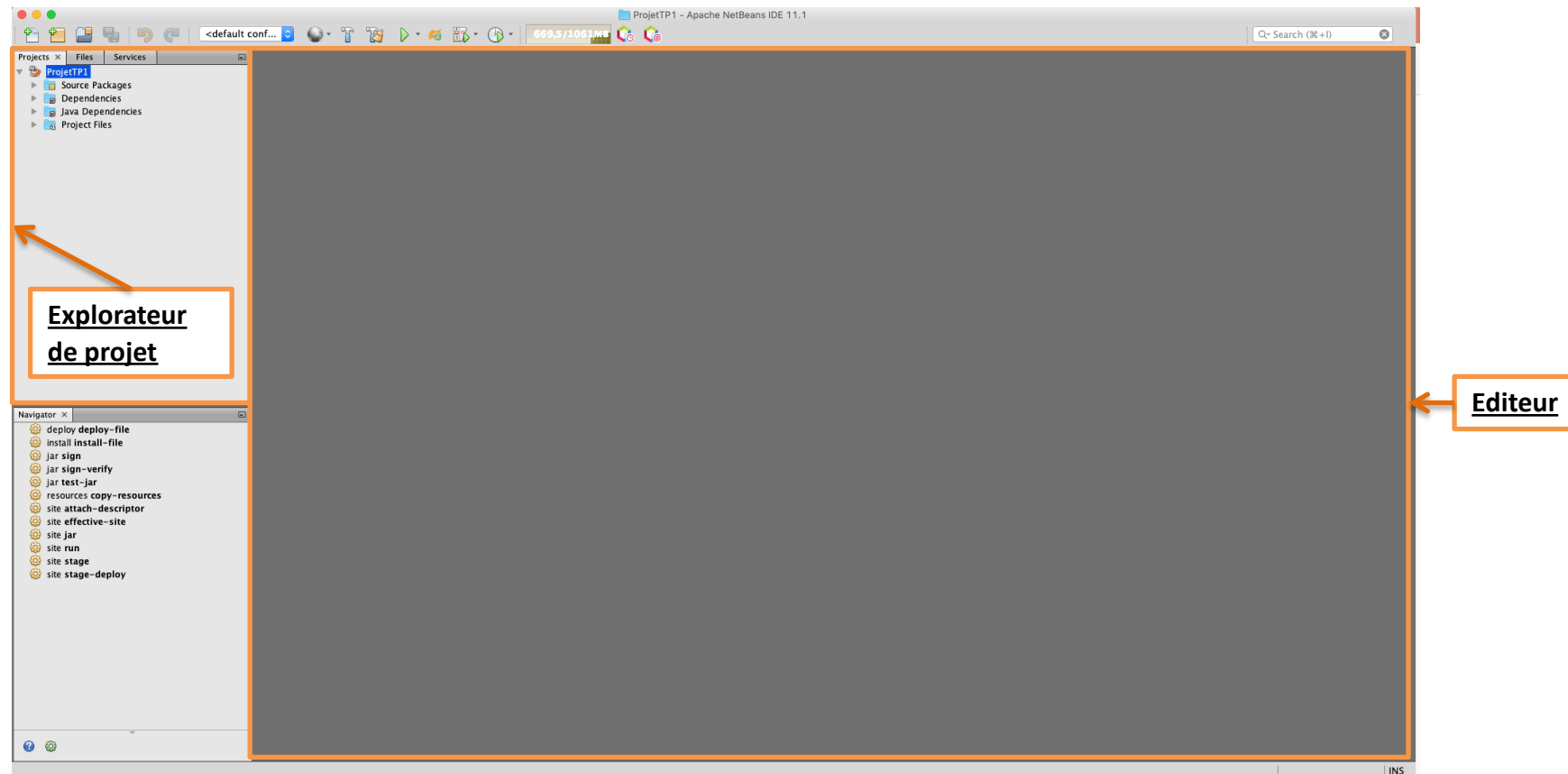
- Project Name:
- Project Location:
- Project Folder:
- Artifact Id:
- Group Id:
- Version:
- Package: (Optional)

At the bottom, there are buttons for 'Help', '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

Création d'un projet (3)

4. **NetBeans** ouvre alors le projet et les fichiers liés dans son éditeur

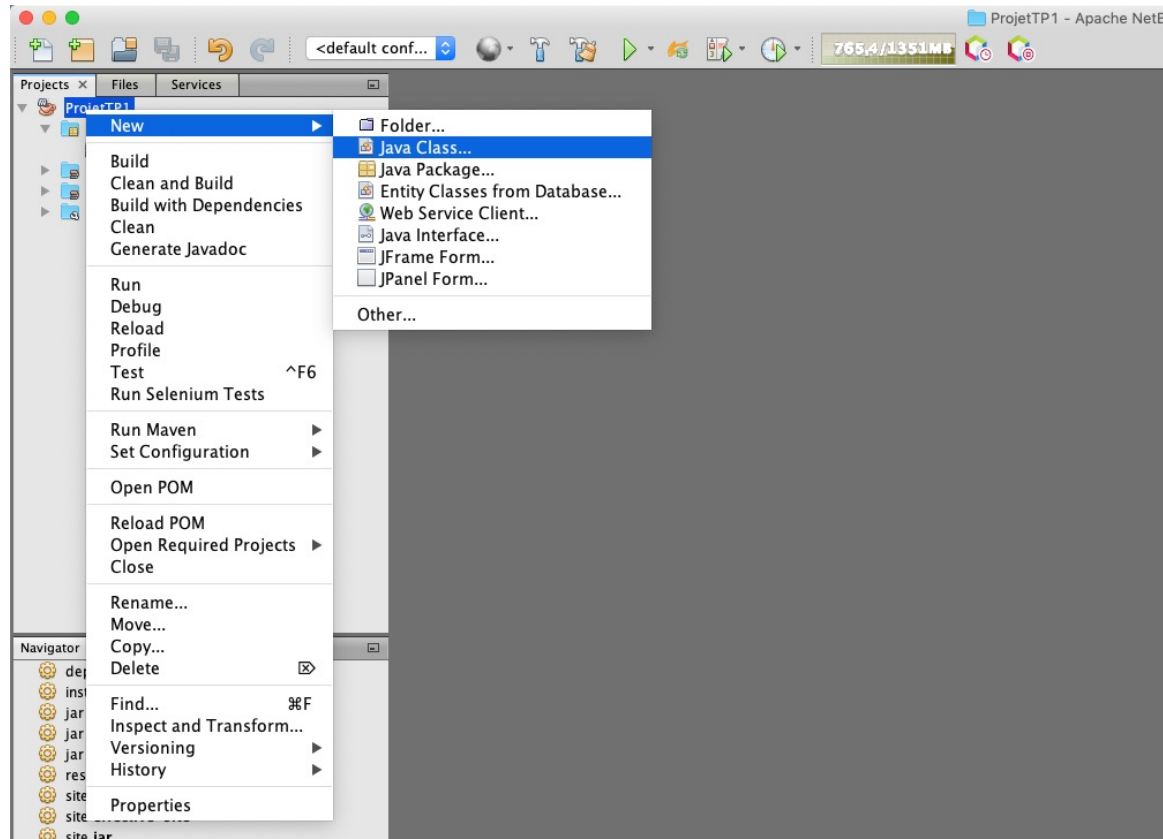
- **Explorateur de Projets** sur la gauche
- **Editeur** de fichiers au centre
- Lorsque l'on rajoutera des fichiers d'autres fenêtres s'ouvriront



Création d'un projet (4)

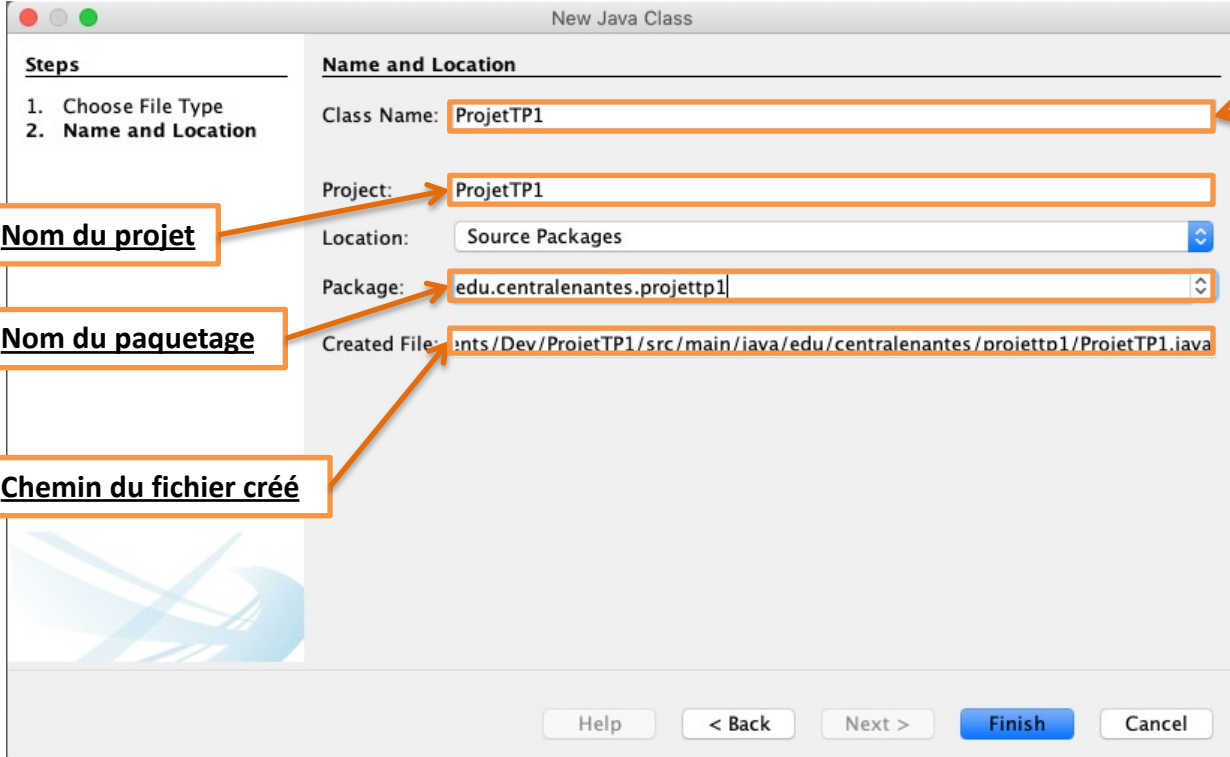
5. Ajouter de nouvelles classes à votre projet

- Pour ce faire, il suffit de faire un clic droit sur le **nom du projet** et de choisir « **New → Java Class** »



Création d'un projet (5)

6. Choisir le **nom de votre classe** et le **paquetage** (package) auquel elle appartiendra
- Les paquetages disponibles (déjà existants) sont proposés par NetBeans via une liste déroulante



The screenshot shows the 'New Java Class' dialog box in NetBeans. The 'Steps' panel on the left indicates the current step is '2. Name and Location'. The 'Name and Location' panel contains the following fields:

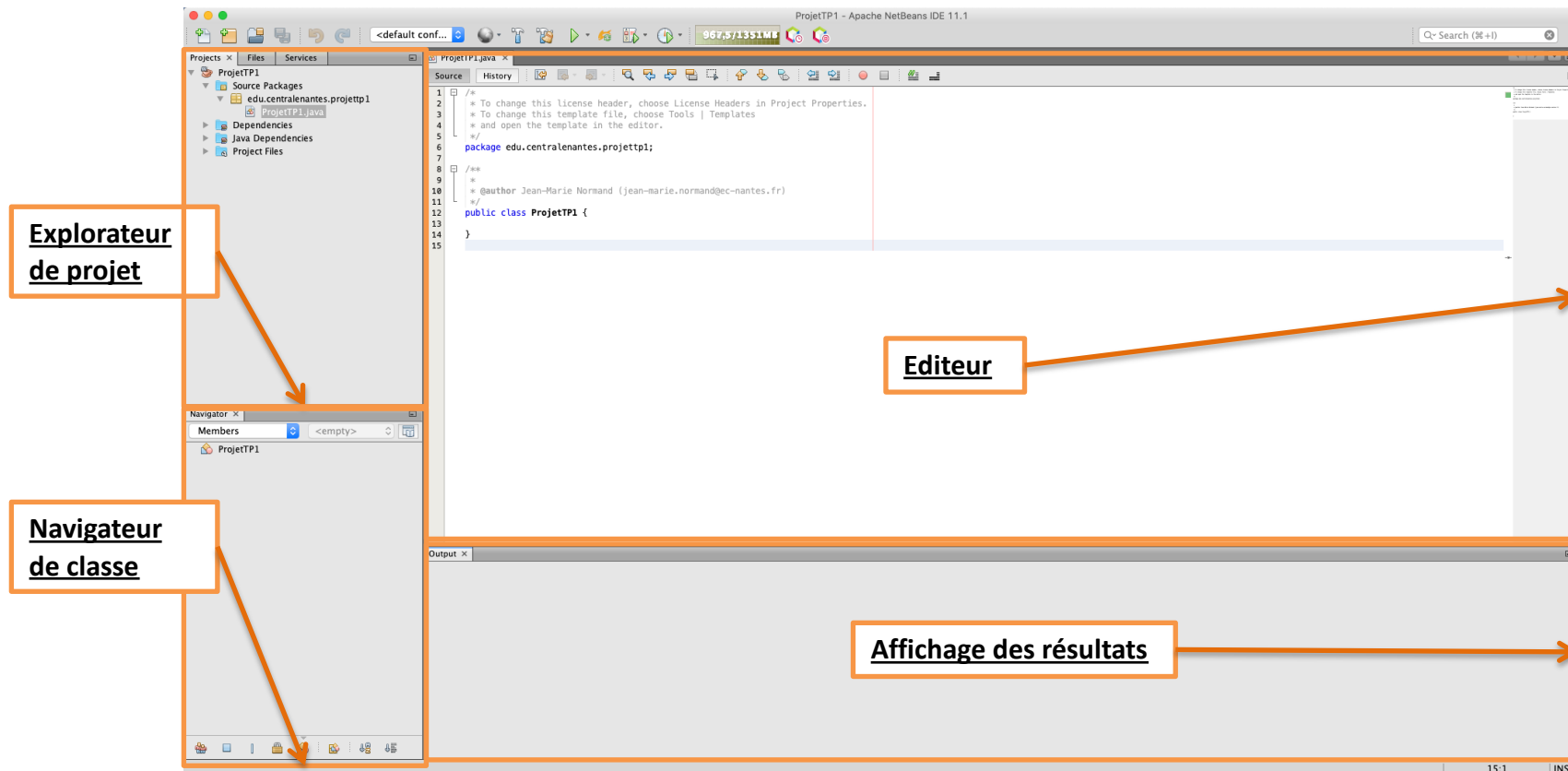
- Class Name:** ProjctTP1 (Annotated with 'Nom de la classe')
- Project:** ProjctTP1 (Annotated with 'Nom du projet')
- Location:** Source Packages
- Package:** edu.centrale Nantes.projcttp1 (Annotated with 'Nom du paquetage')
- Created File:** ...nts/Dev/ProjctTP1/src/main/java/edu/centrale Nantes/projcttp1/ProjctTP1.java (Annotated with 'Chemin du fichier créé')

At the bottom of the dialog are buttons for 'Help', '< Back', 'Next >', 'Finish', and 'Cancel'.

Création d'un projet (6)

7. **NetBeans** affiche alors le contenu de votre fichier dans son éditeur

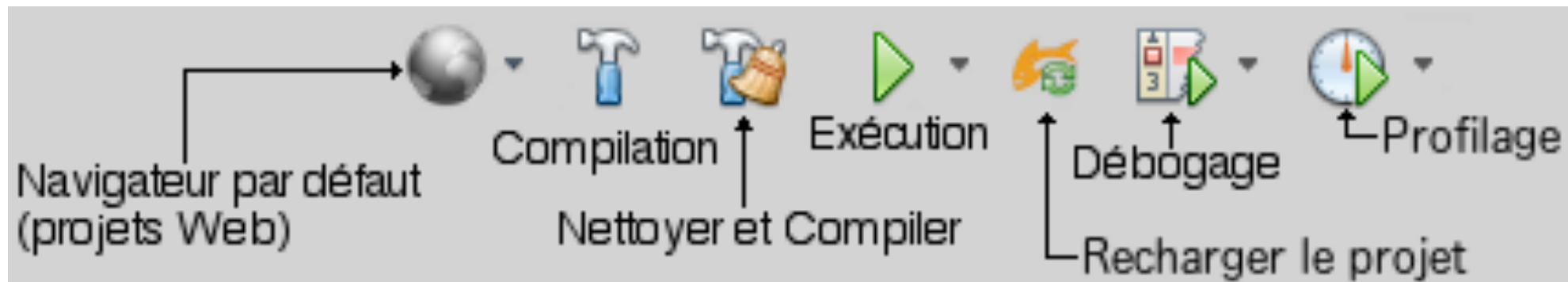
- **Explorateur de Projets** en haut à gauche
- **Navigateur de classe** (ensemble des méthodes et attributs de la classe) en bas à gauche
- **Editeur** de fichiers au centre
- Fenêtre d'**affichage des résultats** de compilation et de la **console** en bas à droite



Création d'un projet (7)

8. Compilation et exécution d'un projet Java

- NetBeans propose un ensemble d'outils pour **compiler**, **exécuter**, **recharger**, **déboguer** et « **profiler** » vos projets
- Ils sont accessibles via les menus éponymes mais aussi directement via les icones suivantes :



Création d'un projet (8)

9. Générer la documentation de votre projet

- Si vous écrivez la **Javadoc** (cf. cours) NetBeans peut générer pour vous la documentation de votre projet. Pour ce faire choisissez le menu : **Run → Generate Javadoc**
- NetBeans génère alors les fichiers HTML au même format que la documentation Java disponible en ligne

10. Ne pas oublier de souvent consulter la Javadoc en ligne !

- <http://docs.oracle.com/javase/8/docs/api/>

11. À vous de jouer !

Structure d'un projet NetBeans (Maven)

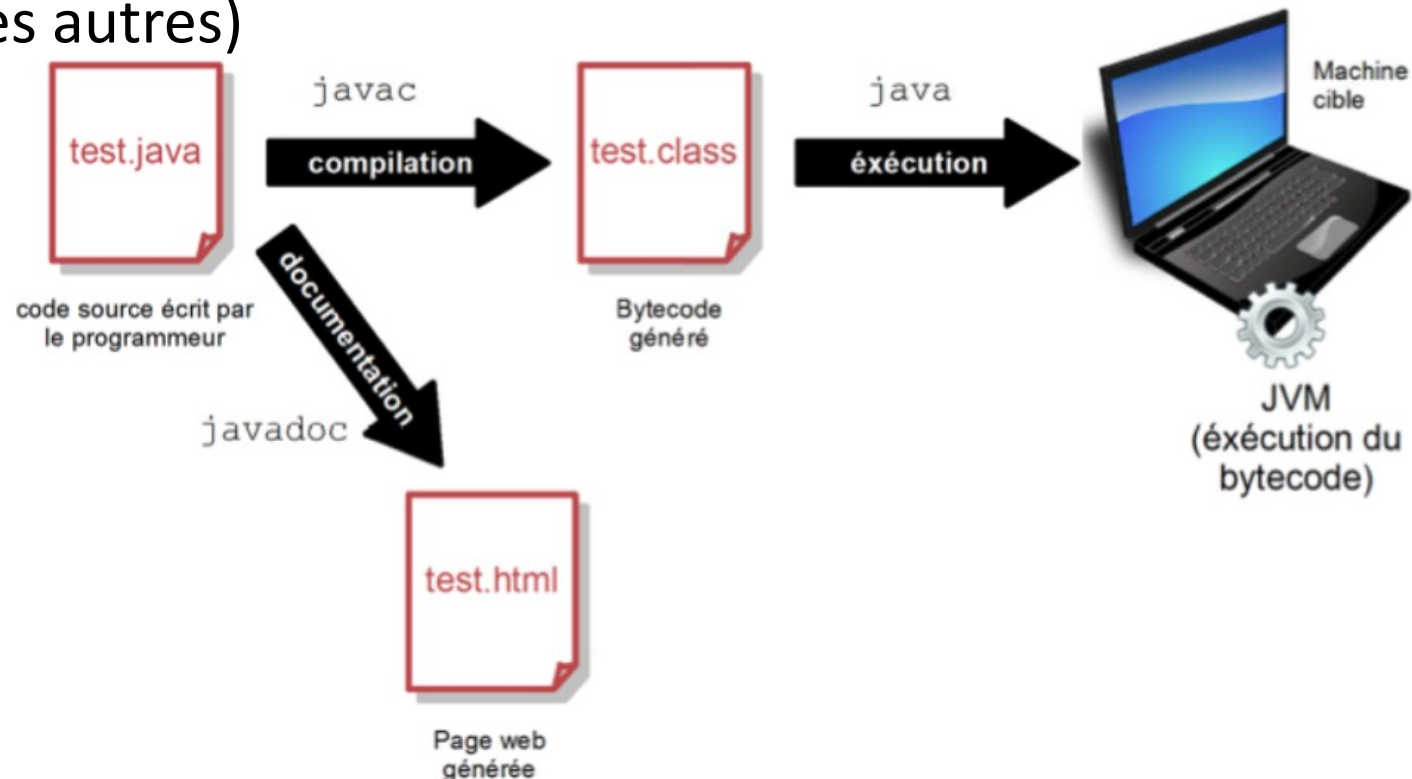
- Un projet = un dossier (ici le dossier racine s'appelle ProjetTP1 on voit son contenu)
- Un dossier **src** qui contient les fichiers Java
 - Lorsque l'on **ajoute un fichier** dans ce dossier il **est automatiquement ajouté au projet** !
 - Lorsque l'on **retire un fichier** de ce dossier il **est automatiquement supprimé du projet** !
 - Un **paquetage (package)** = un sous-dossier de src !
- Un dossier **site** dans lequel un dossier **apidocs** est créé, entre autres, lorsque l'on génère la Javadoc
- Un dossier **target** qui contient les fichiers générés lors de la compilation et de la génération de l'exécutable.
 - Ces fichiers sont stockés dans plusieurs sous dossiers dont :
 - Un sous-dossier **classes** qui contient :
 - Un dossier pour chaque paquetage (ici **projettp1**) qui contient les fichiers compilés (i.e. des **.class**) des fichiers sources de ce dossier (i.e. des **.java**)
- Un fichier **pom.xml** requis par Maven afin de pouvoir générer l'exécutable de votre projet à partir des sources
 - Le fichier contient les paramètres du projet à compiler
 - Apache Maven sert à produire un logiciel à partir de ses sources
- Des dossiers **maven-** qui contiennent des informations nécessaires à Maven

Nom	Date de modification	Taille	Type
pom.xml	aujourd'hui à 14:41	660 octets	XML Document
src	aujourd'hui à 15:43	--	Dossier
main	aujourd'hui à 15:48	--	Dossier
java	aujourd'hui à 15:48	--	Dossier
edu	aujourd'hui à 15:48	--	Dossier
centralenantes	aujourd'hui à 15:48	--	Dossier
projettp1	aujourd'hui à 15:43	--	Dossier
Point2D.java	aujourd'hui à 15:44	4 Ko	TextWr...cument
ProjetTP1.java	aujourd'hui à 15:44	5 Ko	TextWr...cument
target	aujourd'hui à 15:46	--	Dossier
classes	aujourd'hui à 15:45	--	Dossier
edu	aujourd'hui à 15:45	--	Dossier
centralenantes	aujourd'hui à 15:45	--	Dossier
projettp1	aujourd'hui à 15:45	--	Dossier
Point2D.class	aujourd'hui à 15:45	2 Ko	Fichier...se Java
ProjetTP1.class	aujourd'hui à 15:45	2 Ko	Fichier...se Java
generated-sources	aujourd'hui à 15:49	--	Dossier
javadoc-bundle-options	aujourd'hui à 15:46	--	Dossier
maven-archiver	aujourd'hui à 15:45	--	Dossier
maven-status	aujourd'hui à 15:45	--	Dossier
ProjetTP1-0.1.jar	aujourd'hui à 15:45	4 Ko	Fichier JAR Java
site	aujourd'hui à 15:49	--	Dossier
apidocs	aujourd'hui à 15:46	--	Dossier
allclasses-frame.html	aujourd'hui à 15:46	904 octets	HTML
allclasses-noframe.html	aujourd'hui à 15:46	864 octets	HTML
constant-values.html	aujourd'hui à 15:46	4 Ko	HTML
deprecated-list.html	aujourd'hui à 15:46	4 Ko	HTML
edu	aujourd'hui à 15:46	--	Dossier
help-doc.html	aujourd'hui à 15:46	9 Ko	HTML
index-all.html	aujourd'hui à 15:46	10 Ko	HTML
index.html	aujourd'hui à 15:46	3 Ko	HTML
overview-tree.html	aujourd'hui à 15:46	5 Ko	HTML
package-list	aujourd'hui à 15:46	29 octets	TextEdit
script.js	aujourd'hui à 15:46	827 octets	JavaScript
stylesheet.css	aujourd'hui à 15:46	13 Ko	TextWr...cument
test-classes	aujourd'hui à 15:45	--	Dossier

2^E PARTIE : PRISE EN MAIN DE NETBEANS, COMPILATION

Compilation vs. Exécution

- Rappel : il y a une différence entre compilation et exécution d'un programme
- **Compilation** : vérifie que le programme est syntaxiquement correct et transforme chaque fichier **.java** en un fichier **.class** contenant les instructions binaires pour l'exécution
- **Exécution** : exécute les instructions du programme de manière séquentielle (les unes après les autres)



Compilation vs. Exécution

- Rappel bis : **l'ordinateur exécute les instructions que VOUS (ou un autre programmeur) avez écrites**, il n'invente rien, ne veut rien, les phrases du type « l'ordinateur il veut pas » n'ont pas de sens !
- Il existe plusieurs types d'erreurs (en simplifiant) :
 - Les **erreurs syntaxiques** → elles empêchent la **compilation** !
 - Les **erreurs d'exécution** → elles aboutiront soit à :
 - Une **erreur d'exécution** (i.e. un plantage) du programme
 - Un **résultat erroné** de l'exécution du programme




Compilation d'un projet

- Nous allons maintenant illustrer comment compiler un projet et corriger les erreurs éventuelles lors de l'écriture de code Java
- **NetBeans vous aide lors de la compilation** et pour la **correction d'erreurs** !
- **Faites bien attention aux indications données par NetBeans MAIS ne suivez pas toujours aveuglément ses conseils !**
- Il est possible que ce qu'il pense être une solution à une de vos erreurs soit en fait une mauvaise solution !
- **→ Regardez les indications mais ne les suivez pas sans réfléchir !!!**

Compilation d'un projet (2)

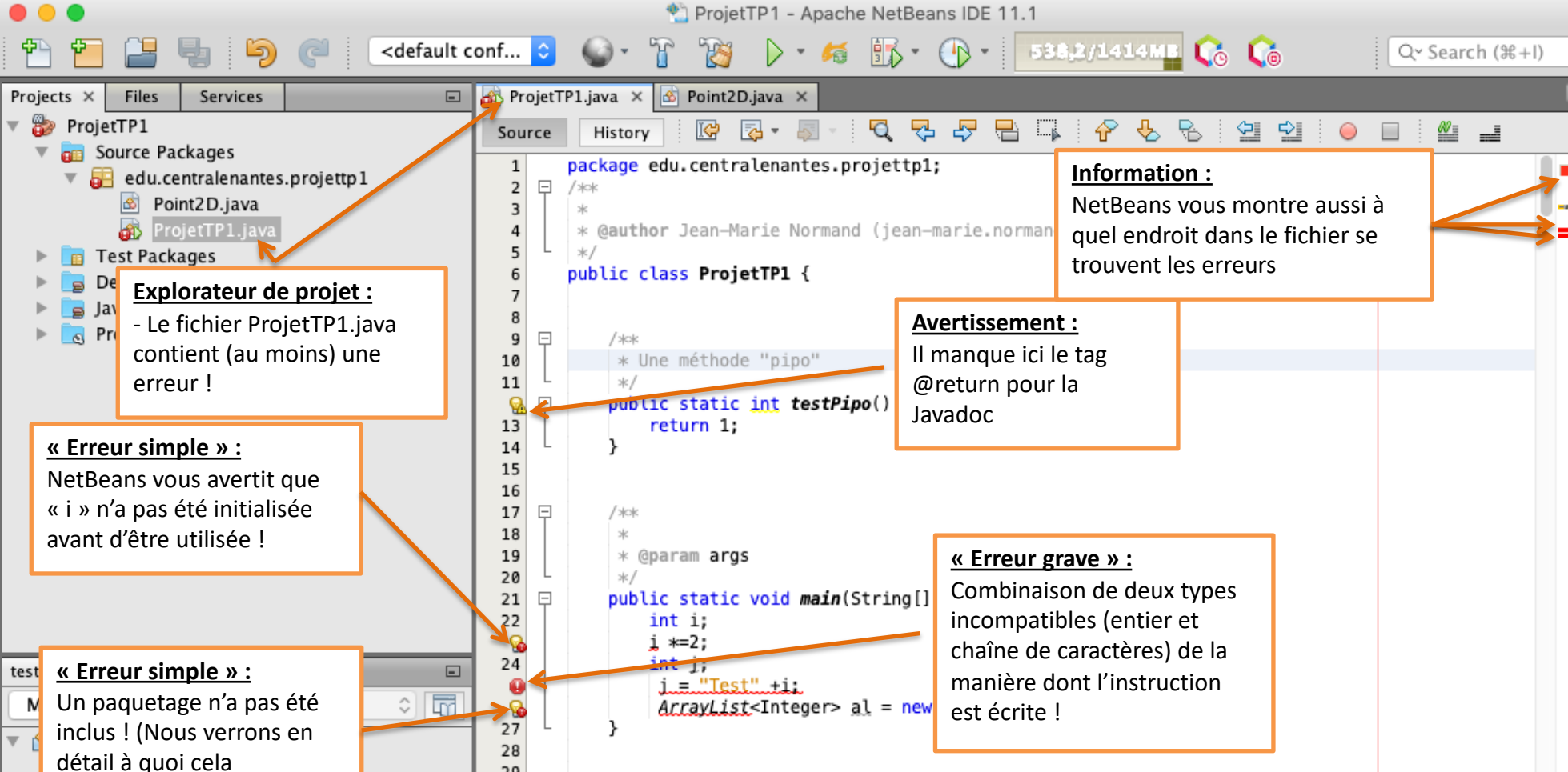
- **NetBeans = Compilation à la volée !!**
- NetBeans vérifie la syntaxe de votre code au fur et à mesure de la saisie
- NetBeans compile votre code dès que possible et a minima à chaque fois que les fichiers source sont sauvegardés
- Pourquoi des boutons de « **Build** » dans l'interface ?
 - L'utilisation du bouton « **Build** » est en théorie inutile
 - Par contre le « **Clean and Build** » peut être utile :
 - En effet, il arrive que des fichiers temporaires créés lors d'une compilation antérieure empêche le projet courant de se compiler (par exemple on a renommé un fichier).
 - Dans ce cas, faire un « **Clean and Build** » vous permet de remettre le projet dans un état stable et correct !

Compilation d'un projet (3)

- **NetBeans = Compilation à la volée !!**
- NetBeans compile votre code à chaque fois que les fichiers source sont sauvegardés
- Différents types d'indications de NetBeans sur l'état de votre code :
 -  – Un **avertissement** (warning) : votre attention est attirée sur une ligne de code qui ne génère pas d'erreur de compilation mais sur laquelle on vous propose une amélioration
 -  – Une **erreur « simple »** (réelle ou potentielle) que NetBeans sait résoudre et sur laquelle il va vous proposer une solution
 -  – Une **erreur « grave »** qui empêche la compilation et que NetBeans ne sait pas résoudre : aucune proposition de résolution ne sera proposée
- Chaque type d'erreur possède une icône particulière et en laissant la souris sur l'icône on voit la proposition de NetBeans et l'on peut l'accepter ou non
- **Dans l'explorateur de projet** : on voit également les fichiers contenant des erreurs !

Compilation d'un projet (4)

- Exemple :



Explorateur de projet :
- Le fichier ProjetTP1.java contient (au moins) une erreur !

« Erreur simple » :
NetBeans vous avertit que « i » n'a pas été initialisée avant d'être utilisée !

« Erreur simple » :
Un paquetage n'a pas été inclus ! (Nous verrons en détail à quoi cela correspond plus tard)

Information :
NetBeans vous montre aussi à quel endroit dans le fichier se trouvent les erreurs

Avertissement :
Il manque ici le tag @return pour la Javadoc

« Erreur grave » :
Combinaison de deux types incompatibles (entier et chaîne de caractères) de la manière dont l'instruction est écrite !

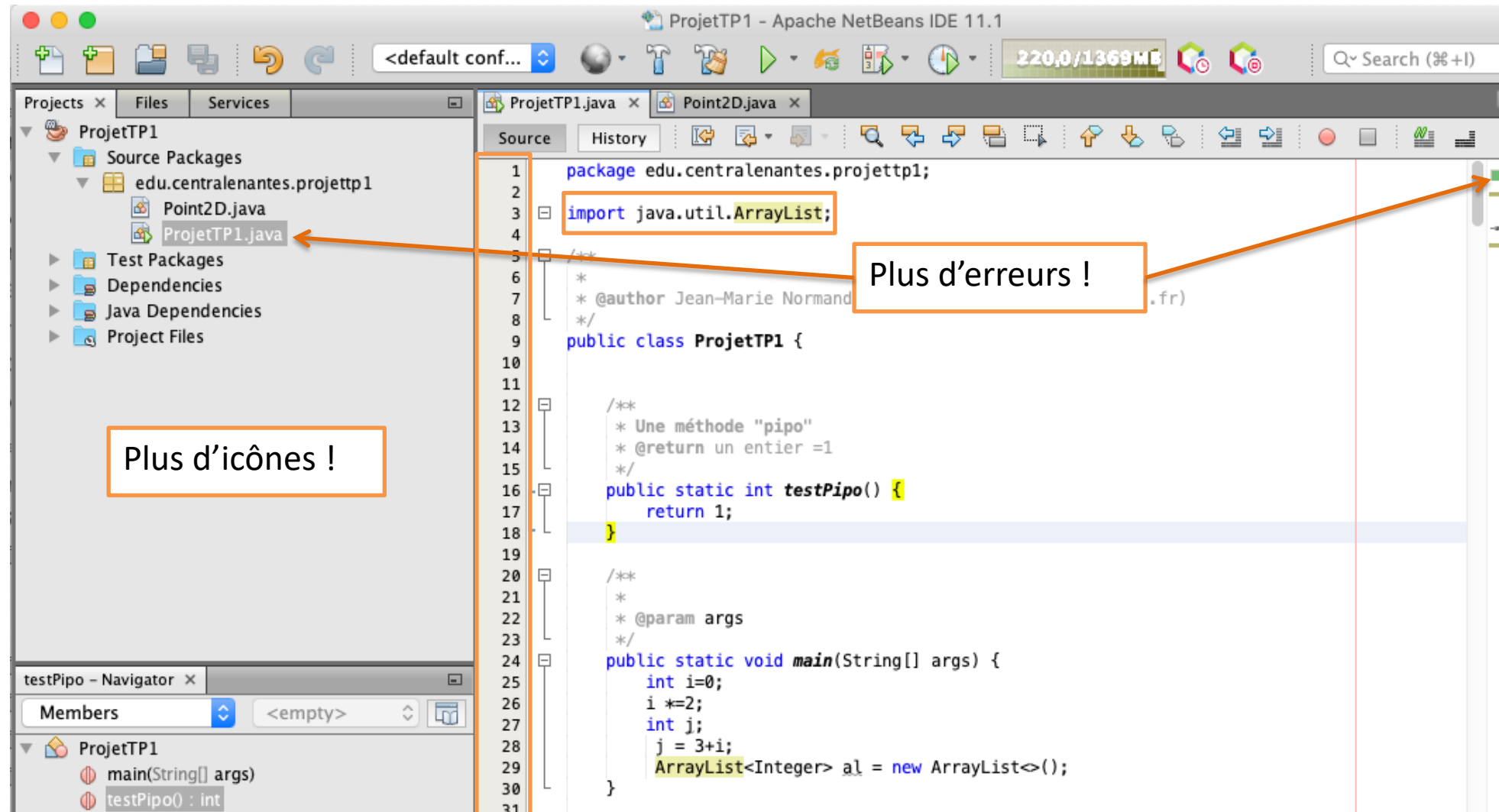
```

1 package edu.centralenantes.projett1;
2 /**
3  *
4  * @author Jean-Marie Normand (jean-marie.normand)
5  */
6 public class ProjetTP1 {
7
8
9     /**
10      * Une méthode "pipo"
11      */
12     public static int testPipo()
13     {
14         return 1;
15     }
16
17     /**
18      *
19      * @param args
20      */
21     public static void main(String[] args) {
22         int i;
23         i *= 2;
24         int j;
25         j = "Test" + i;
26         ArrayList<Integer> al = new ArrayList<>();
27     }
28
29 }

```

Compilation d'un projet (5)

- Les erreurs corrigées : le projet compile !

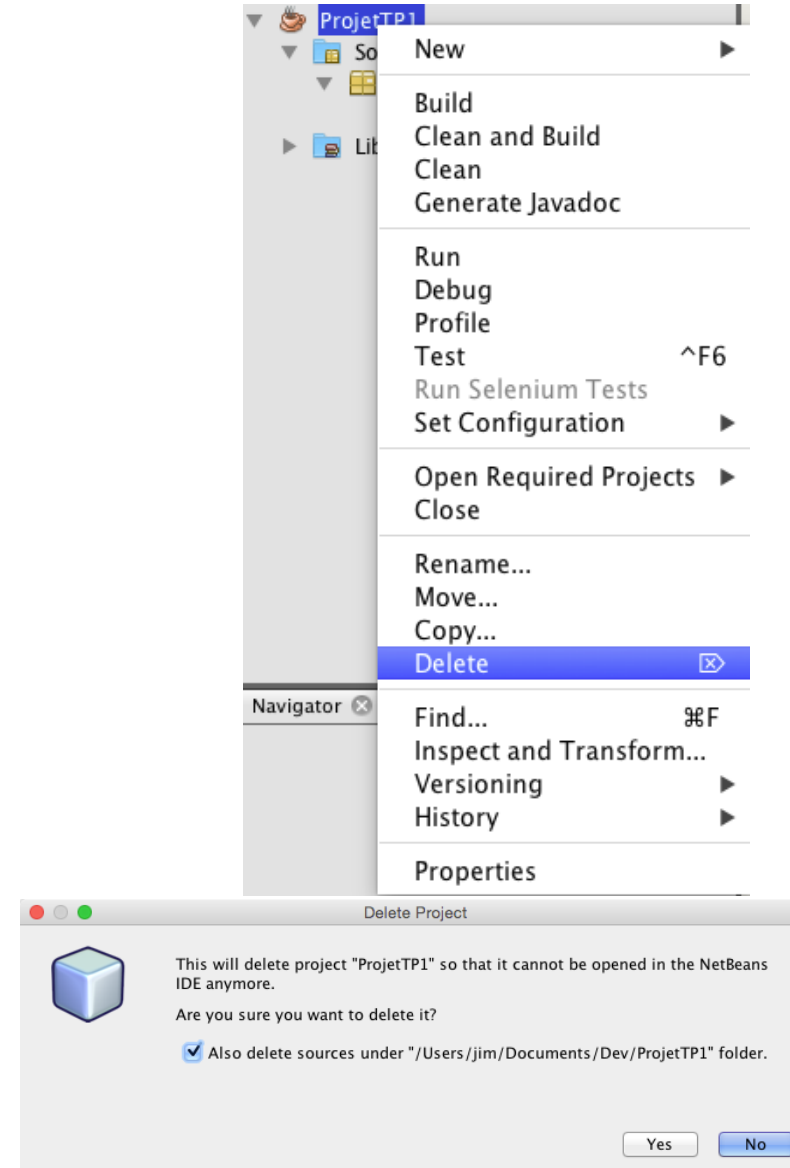


Prise en main de l'éditeur

- Pour plus d'informations sur l'éditeur de NetBeans, allez voir les ressources en ligne :
 - <https://netbeans.org/kb/docs/java/editor-codereference.html>
 - http://docs.oracle.com/cd/E50453_01/doc.80/e50452/working_nbeans.htm
- Vous apprendrez en utilisant l'IDE au fur et à mesure de vos développements


Suppression d'un projet

- Si vous souhaitez **supprimer un projet**, vous pouvez le faire depuis l'interface de NetBeans
- Pour ce faire :
 - Cliquez droit sur le nom du projet à supprimer et sélectionnez « **Supprimer** »
 - Sélectionnez « Supprimer également les sources » puis cliquez sur « Oui »





Exercice 1

- Allez sur le serveur pédagogique, dans le dossier du cours d'objet et récupérez le fichier `ProjetTP1-Ex1.java`
- Ce fichier contient du code Java volontairement **syntactiquement et fonctionnellement incorrect** !
- Instructions :
 - Supprimez le projet `ProjetTP1` si vous l'avez créé précédemment (voir slide précédent)
 - Créez un nouveau projet Java dans NetBeans : `ProjetTP1`
 - Intégrez le code contenu dans le fichier `ProjetTP1-Ex1.java`
 - Corrigez les erreurs de manière à ce que le projet compile !
 - Il ne doit rester aucune erreur (simple ou grave) ni aucun avertissement (warning)
 - Dans votre rapport : **listez les modifications** apportées au code pour **qu'il compile** !
 - Expliquez **pourquoi** vous avez fait ces modifications : pourquoi c'était faux et comment vous l'avez corrigé
- Lancez le programme (icône )
- Le code fournit un résultat faux ! → Nous allons voir maintenant comment déboguer un programme c'est-à-dire corriger les erreurs d'exécution de celui-ci

Débogage

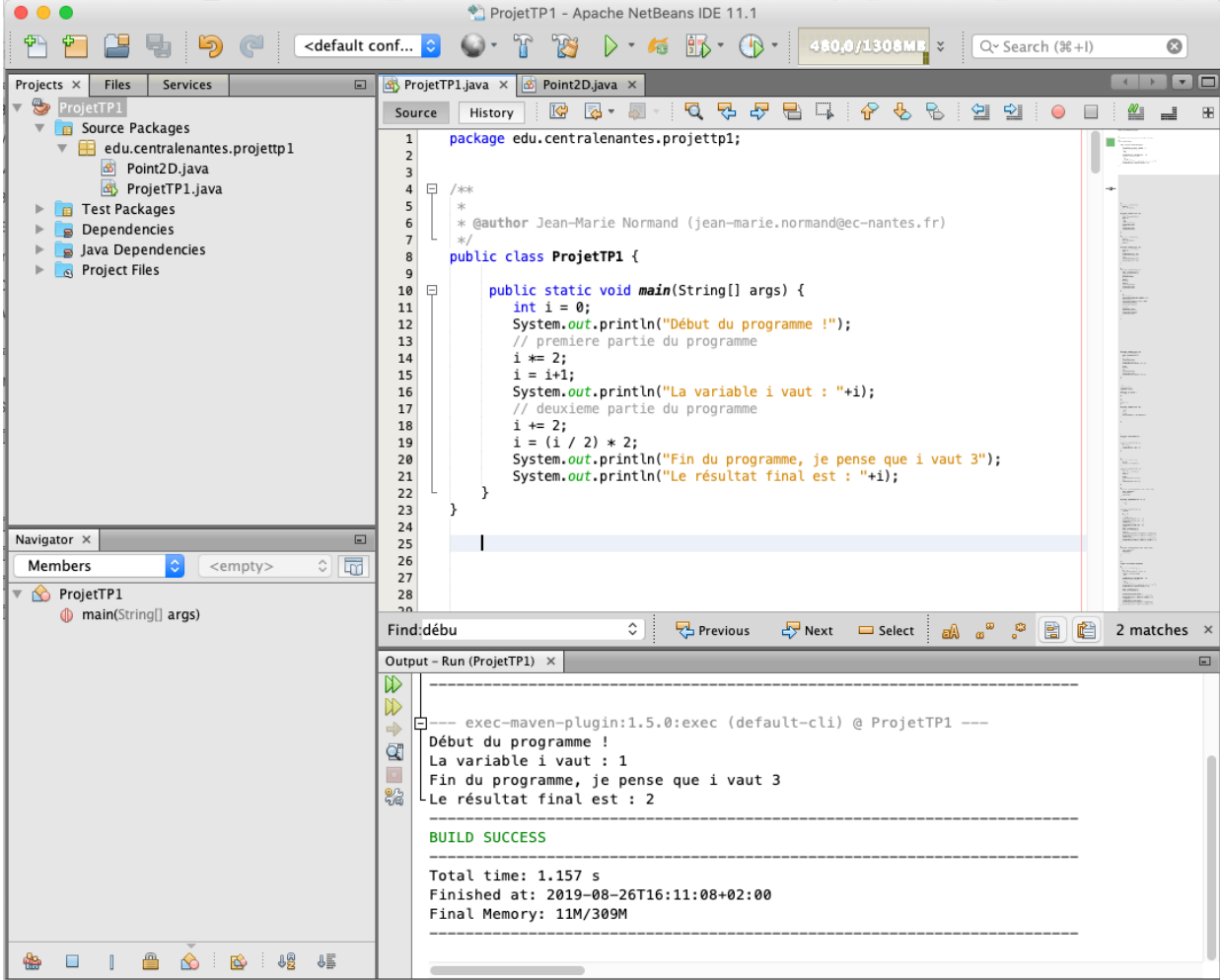
- Le concept de **débogage** consiste à exécuter un programme d'une manière spécifique à la détection et à la correction des erreurs qu'il contient
- On distinguera dans la suite deux grands types de débogage :
 1. Le **débogage « à la main »** : aussi appelé débogage au « println »
 2. Le **débogage « pas à pas »** : qui tire profit des débogueurs intégrés aux IDE

Débogage (2)

- Le **débogage « à la main »** : version simple voire simpliste mais qui permet néanmoins de trouver des erreurs !
- Idée : mettre des **affichages textuels** dans la console (`System.out.println` en Java) afin de vérifier ce qui se passe pendant l'exécution du programme
- Si le **programme plante** → cela permet de trouver d'où vient l'erreur en procédant par dichotomie :
 - Un affichage au début (message **m1**) du programme puis un à la fin (**m2**)
 - Si l'on ne voit pas **m2**, alors on rajoute un troisième message (**m3**) à peu près au milieu du code et on relance l'exécution
 - Si l'on voit **m3** alors on réitère en rajoutant **m4** entre **m3** et **m2**
 - Sinon : on réitère en rajoutant **m4** entre **m1** et **m3**
 - Etc.
- Si le **résultat est inexact** → on procède de la même manière mais en affichant les valeurs des variables qui nous intéressent
- Ainsi on peut vérifier quelle(s) instruction(s) provoquent l'(es) erreur(s)

Débogage (3)

- Le **débogage « à la main »** : version simple voire simpliste mais qui permet néanmoins de trouver des erreurs !
- Rappel : affichage textuel en Java
→ `System.out.println("chaîne de caractères")`
- Java nous permet de transformer les types de base en chaîne de caractères pour faire des affichages plus complets



```
package edu.centralnantes.projett1;

/**
 * @author Jean-Marie Normand (jean-marie.normand@ec-nantes.fr)
 */
public class ProjetTP1 {

    public static void main(String[] args) {
        int i = 0;
        System.out.println("Début du programme !");
        // première partie du programme
        i += 2;
        i = i+1;
        System.out.println("La variable i vaut : "+i);
        // deuxième partie du programme
        i += 2;
        i = (i / 2) * 2;
        System.out.println("Fin du programme, je pense que i vaut 3");
        System.out.println("Le résultat final est : "+i);
    }
}
```

Find:débu 2 matches

Output - Run (ProjetTP1)

```
exec-maven-plugin:1.5.0:exec (default-cli) @ ProjetTP1 ---
Début du programme !
La variable i vaut : 1
Fin du programme, je pense que i vaut 3
Le résultat final est : 2

BUILD SUCCESS

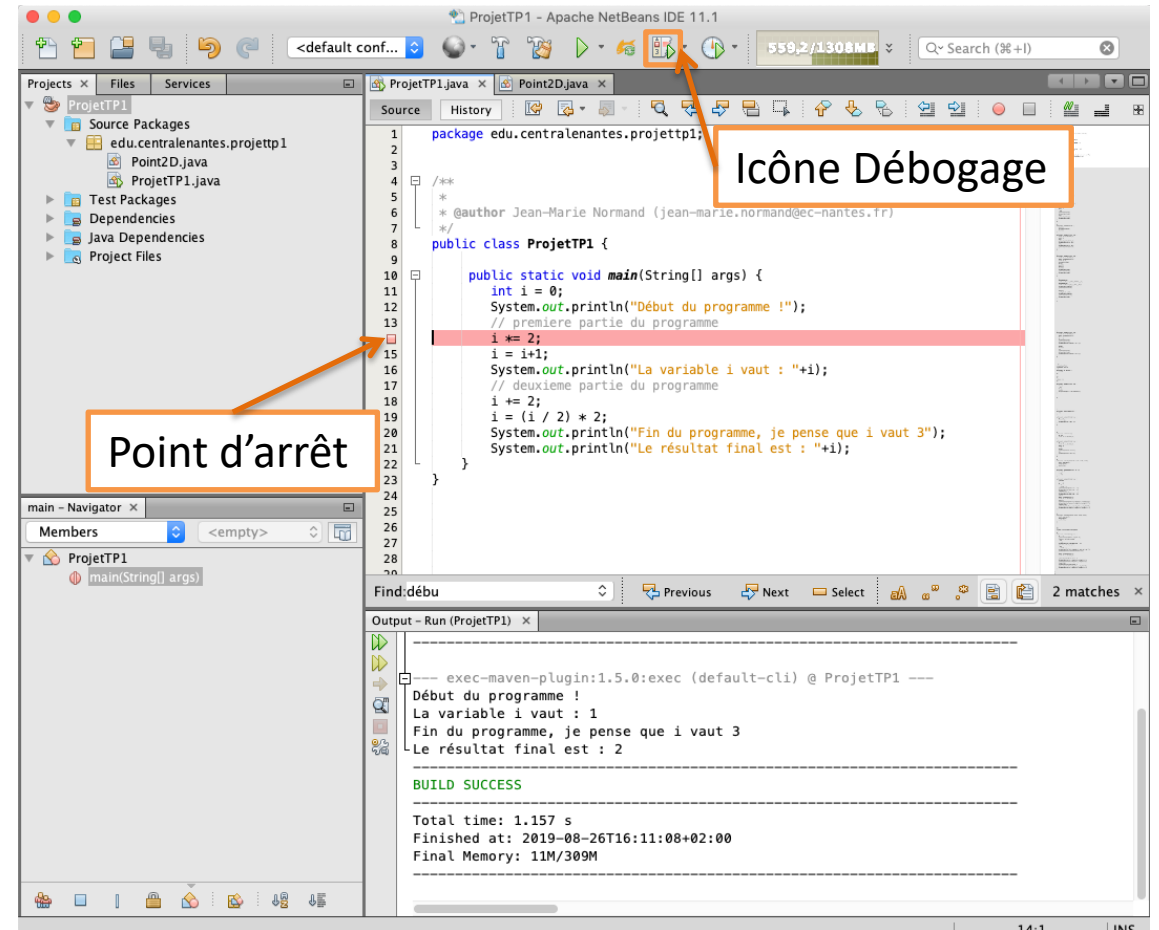
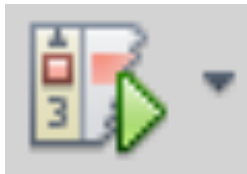
Total time: 1.157 s
Finished at: 2019-08-26T16:11:08+02:00
Final Memory: 11M/309M
```

Débogage (4)

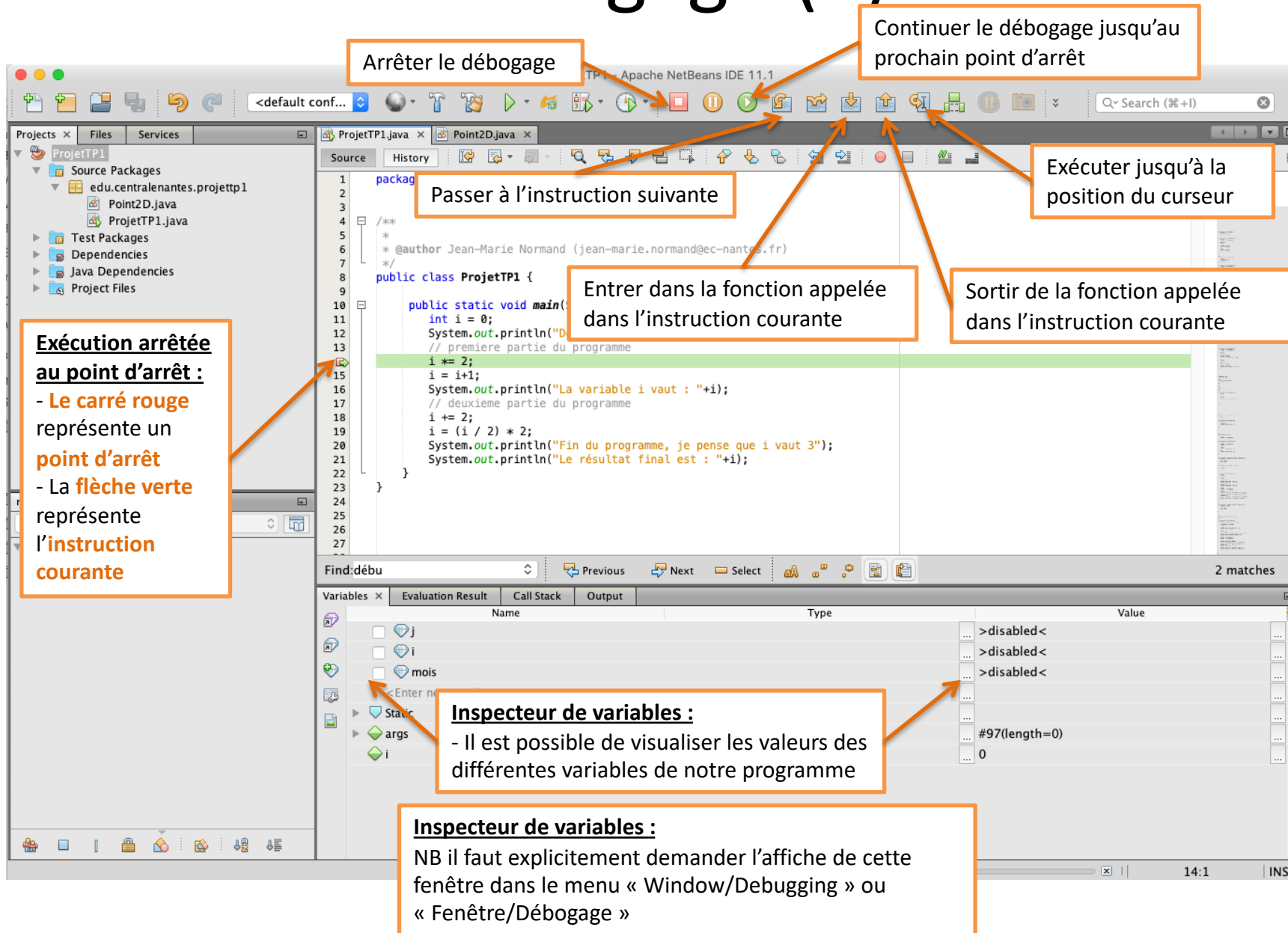
- Le **débogage « pas à pas »** : l'utilisateur ajoute dans NetBeans des **points d'arrêts** (*breakpoints*)
- Un point d'arrêt s'ajoute en **cliquant** (clic gauche) **dans la colonne où sont affichés les numéros de lignes** :
 - un carré rouge apparaît alors !
 - La **ligne entière est alors affichée en rouge** dans l'éditeur
 - Si **l'on clique sur le carré rouge** : le point d'arrêt est supprimé
- Lorsque l'exécution du programme arrive à l'instruction du point d'arrêt :
 - **l'exécution s'arrête** et l'on peut regarder le contenu des variables etc.
 - On peut également faire avancer le programme **pas à pas, décider de continuer l'exécution jusqu'au prochain point d'arrêt**, etc.
- Ce mode de débogage nous permet d'avoir un contrôle très fin des instructions et ainsi de détecter facilement quelle(s) instruction(s) provoque(nt) l'(es) erreur(s) !!!

Débogage (5)

- **Attention !**
- Pour que le programme s'arrête bel et bien sur les points d'arrêt, il faut lancer le programme en mode **DEBUG** !
- Il faut cliquer sur l'icône « Débogage »



Débogage (6)



Arrêter le débogage

Continuer le débogage jusqu'au prochain point d'arrêt

Exécuter jusqu'à la position du curseur

Sortir de la fonction appelée dans l'instruction courante

Entrer dans la fonction appelée dans l'instruction courante

Passer à l'instruction suivante

Exécution arrêtée au point d'arrêt :

- Le **carré rouge** représente un **point d'arrêt**
- La **flèche verte** représente l'**instruction courante**

Inspecteur de variables :

- Il est possible de visualiser les valeurs des différentes variables de notre programme

Inspecteur de variables :

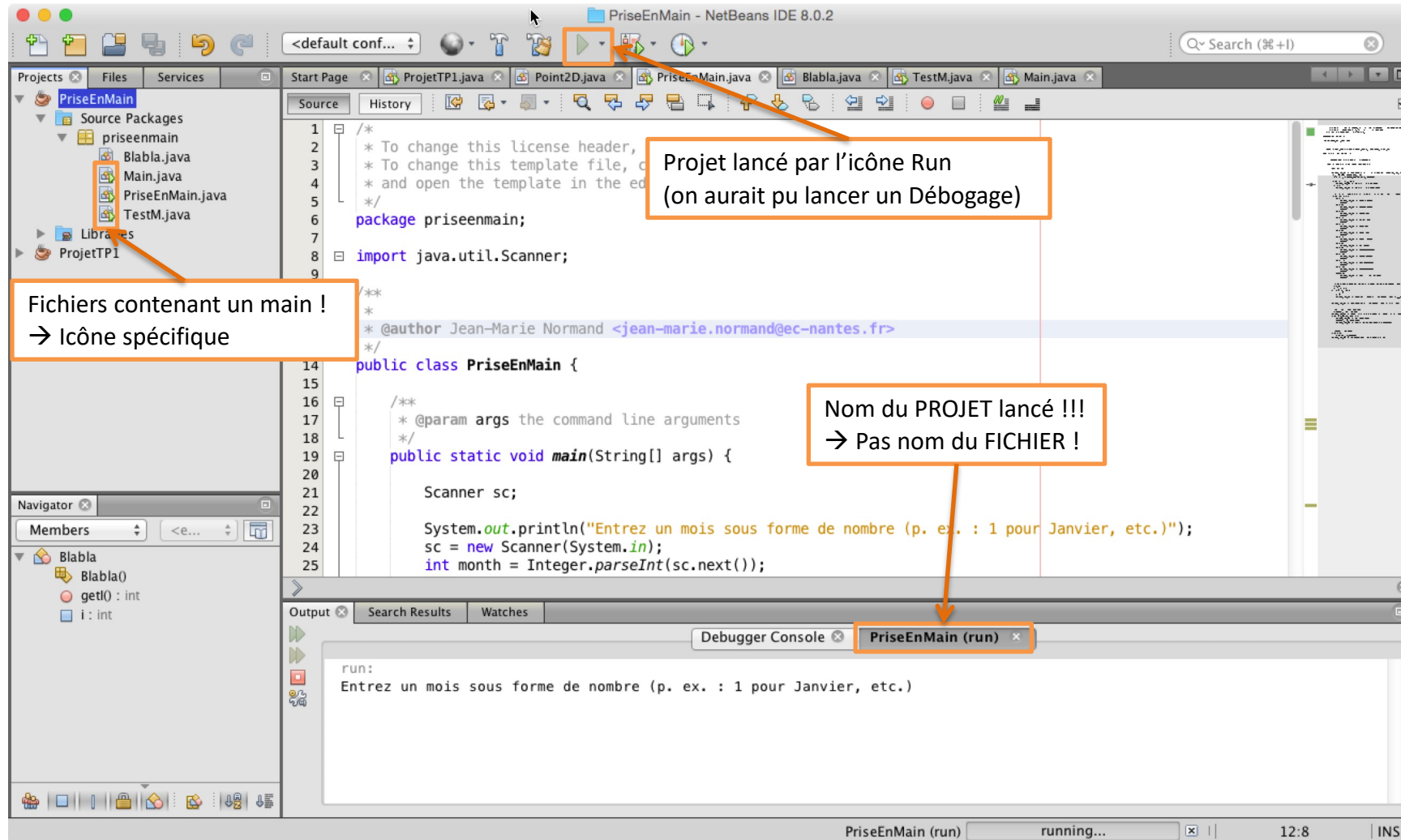
NB il faut explicitement demander l'affiche de cette fenêtre dans le menu « Window/Debugging » ou « Fenêtre/Débugage »

Name	Type	Value
j		>disabled<
i		>disabled<
mois		>disabled<
args		#97(length=0)
i		0

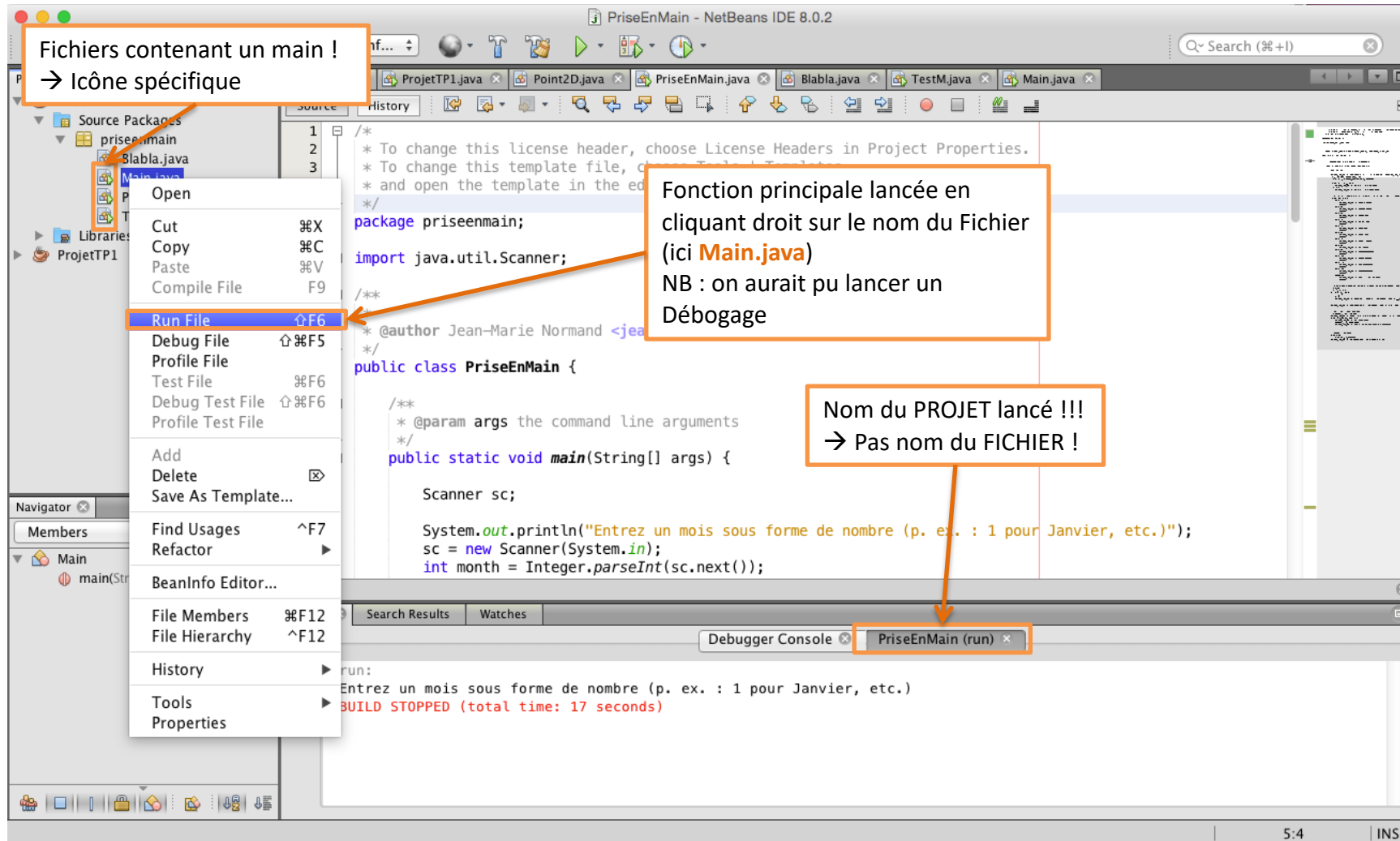
NetBeans : lancer un programme ou un débogage ?

- Rappel Java : dans un projet Java il peut y avoir plusieurs classes contenant une fonction principale (« **main** ») → ces fichiers sont repérables dans NetBeans par leur icône spéciale
- Dans NetBeans lorsque l'on lance le programme ou le débogage alors la fonction **main** lancée dépend du fichier sélectionné !
- Attention car cela peut vous jouer des tours !

NetBeans : lancer un programme ou un débogage ? (2)

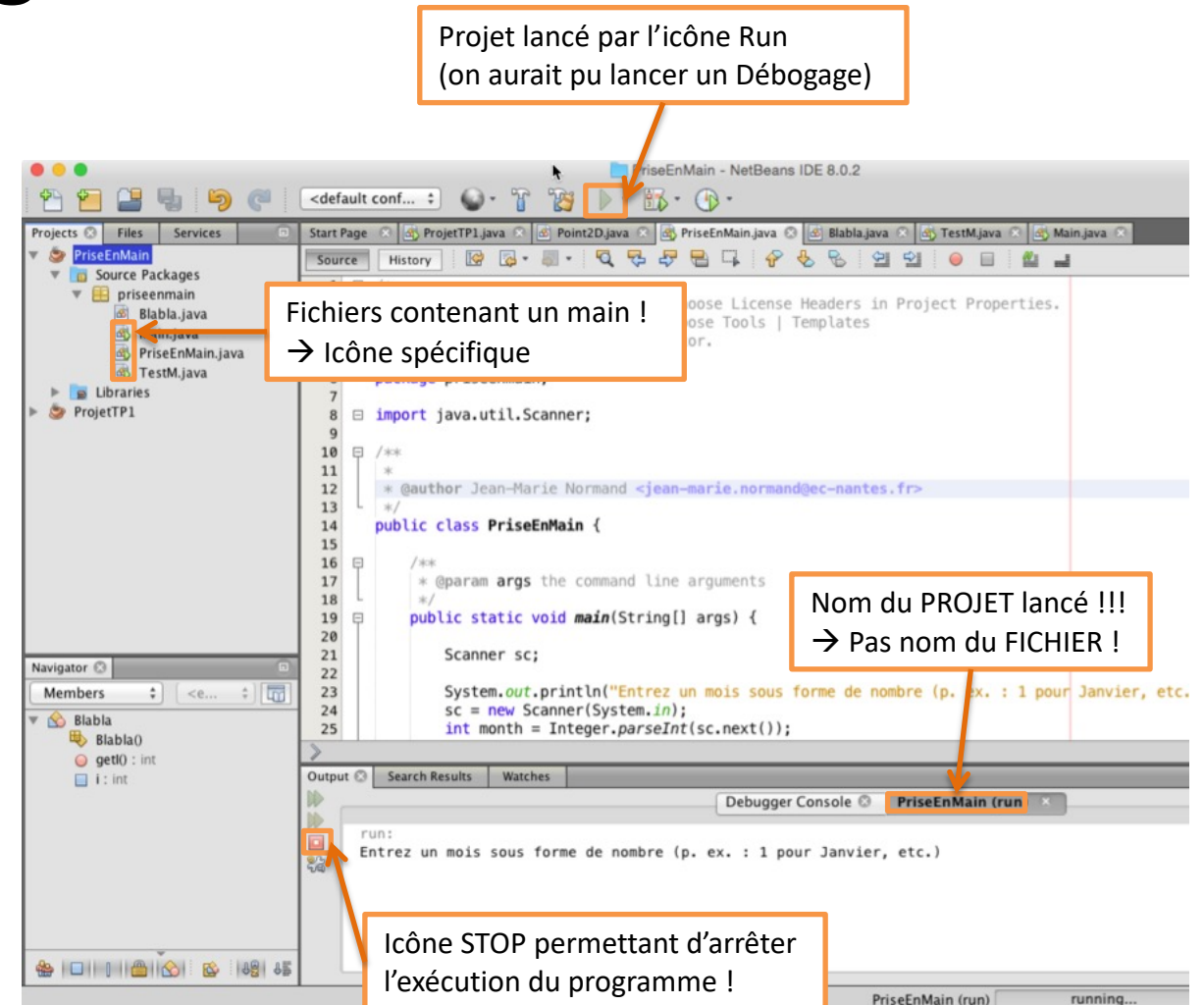


NetBeans : lancer un programme ou un débogage ? (3)



NetBeans : terminer un programme ou un débogage ?

- Il peut être utile **d'arrêter un programme** ou un débogage lancé dans NetBeans, pour plusieurs raisons :
 - Boucle infinies (le programme ne peut s'arrêter)
 - Vous avez trouvé un problème à corriger
 - Etc.





Exercice 1

- Illustrez les modifications que vous avez apportées au code pour que celui-ci compile
- Vérifiez que votre code fonctionne !
- L'année **2000 est-elle bissextile ?**
- L'année **1997 est-elle bissextile ?**
- L'année **1996 est-elle bissextile ?**
- En fonction de ces réponses, le code est-il correct ? Si non, modifiez le et **expliquez ce qui n'était pas correct**
- Dans votre rapport vous **répondrez** également à cette question : **« un code qui compile est-il forcément un code qui fonctionne (qui est correct) ? »**
- Aide sur les années bissextiles ☺ : <http://www.jetcityorange.com/leap-year/french.html>



Exercice 2

- Allez sur le serveur pédagogique, dans le dossier du cours d'objet et récupérez les fichiers `ProjetTP1-Ex2.java` et `Compteur.java`
- Ces fichiers contiennent du code Java volontairement **faux** !
- Instructions :
 - Créez un nouveau projet Java dans NetBeans nommé `PriseEnMain`
 - Intégrez le code contenu dans le fichier `ProjetTP1-Ex2.java` dans la classe principale et ajoutez le fichier `Compteur.java` au projet
 - Corrigez les erreurs de manière à ce que le projet compile !
 - Il ne doit rester aucune erreur (simple ou grave) ni aucun avertissement (warning)
 - Dans votre rapport : **listez les modifications** apportées au code pour **qu'il compile** !
 - Expliquez **pourquoi** vous avez fait ces modifications : pourquoi c'était faux et comment vous l'avez corrigé
 - Utilisez le débogueur pour corriger les fautes et problèmes du programme
 - Dans le rapport : **Illustrez votre utilisation du débogueur** (captures d'écran) et **expliquez les modifications apportées** au code



Exercice 2 (suite)

- Objectif : vous faire comprendre la différence entre un fichier .java et un fichier .class !
- Instructions :
 - D'après la diapositive 12 trouvez les répertoires contenant les fichiers **.class** de votre projet **PriseEnMain**
 - Essayez d'ouvrir le fichier **PriseEnMain.class** dans un éditeur de texte (celui de NetBeans ou un autre de votre choix)
 - Dans votre rapport :
 - **décrivez le contenu de ce fichier !**
 - d'après la diapositive 14 et le cours expliquez **pourquoi** vous voyez ce contenu et à quoi il correspond
 - **en le justifiant**, dites si vous pensez qu'il est plus logique de rendre un fichier de type **.java** ou de type **.class** lors d'un examen ou d'un rendu de projet



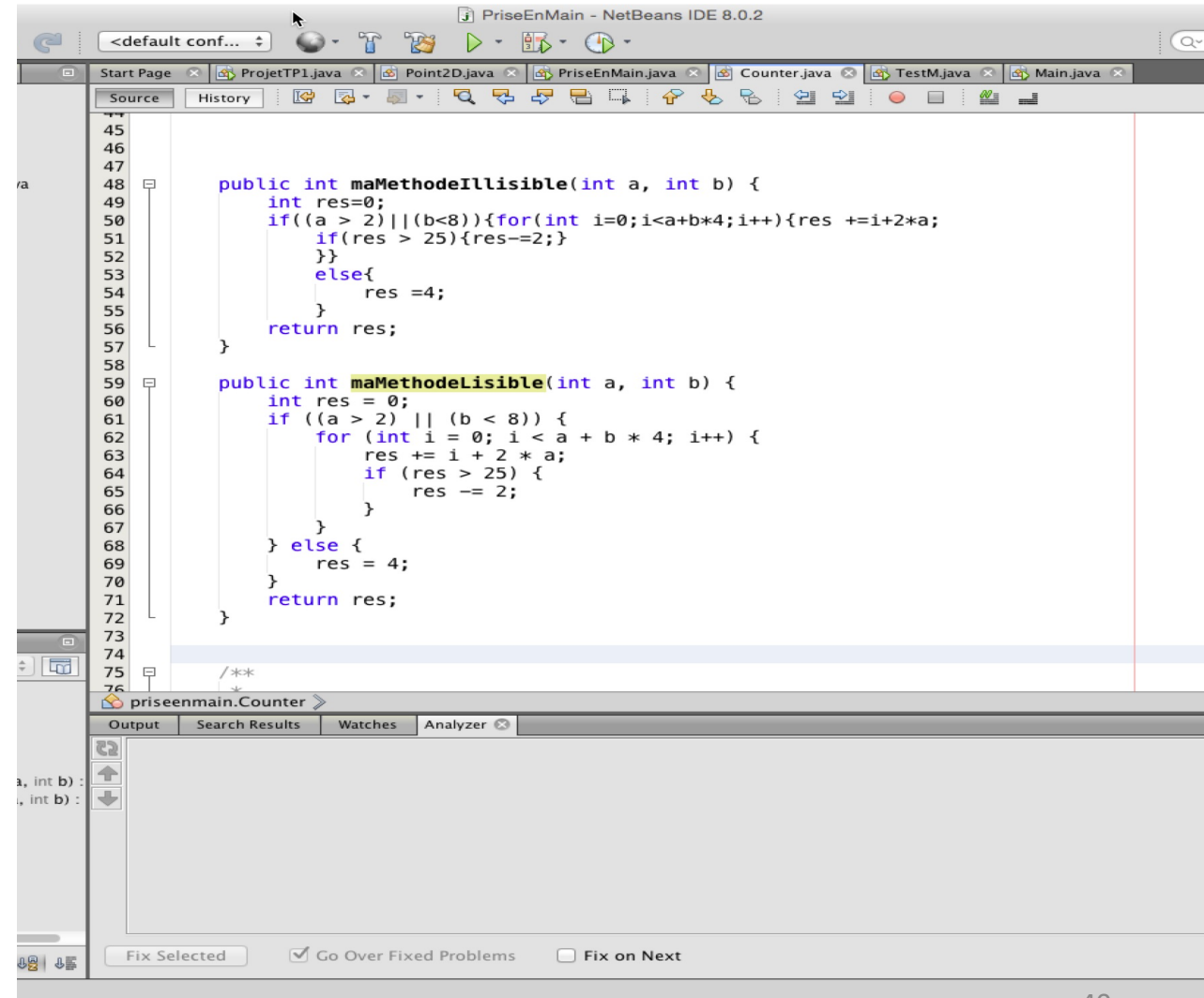
Intérêt du Débogueur ?

- Illustrez des situations/exemples où l'utilisation d'un débogueur vous apparaît intéressante pour un développeur
 - Justifiez pourquoi c'est intéressant/important à vos yeux
 - Vous pouvez si vous le souhaitez donner des captures d'écran de code et/ou du débogueur NetBeans

3^E PARTIE : MA PREMIÈRE CLASSE JAVA : UN POINT EN 2 DIMENSIONS

NetBeans : les incontournables !

- Dernières précisions sur l'utilisation de NetBeans pour coder en Java
- **Mettez votre code en forme !!!!!!!!!!!**
 - Sélectionnez le morceau de code :
 - **Clic droit « Format »** ou
 - Menu « **Source → Format** »
 - Il est également possible de mettre en forme l'ensemble du fichier en une fois
- NetBeans peut **générer du code pour vous** :
 - les accesseurs et modificateurs (qui sont pénibles à écrire), constructeurs, etc.
 - Menu « **Source → Insert Code** » ou **clic droit « Insert Code »**



Une Classe Point2D

- Le but ici est d'implémenter votre première classe : **Point2D**
- Cette classe doit représenter un **point** à **coordonnées entières** dans un plan Cartésien à **deux dimensions**
- Rappel sur les classes en Java :
 - Une classe représente à la fois :
 - L'état interne, le « quoi », d'un objet de cette classe → **les attributs**
 - Le comportement d'un objet de cette classe, ce qu'il est capable de faire, le « comment » → **les méthodes**
 - Une classe doit posséder **un ou plusieurs constructeur(s)**
 - Une classe doit fournir des **accesseurs** et des **modificateurs** sur ses **attributs**
 - Une classe doit fournir des **méthodes** qui vont permettre aux programmeurs-utilisateurs de **manipuler des objets** de cette classe
- Reportez vous au cours pour vérifier les détails et la syntaxe utilisée en Java



Exercice 3

- Créez un nouveau projet Java dans NetBeans, appelez le p. ex. « **Projet** »
- Créez une classe Java **Point2D** dans un paquetage « **org.centrale.projet.objet** » :
 - Deux attributs + accesseurs et modificateurs
 - Comportement demandé à un objet de type Point2D :
 - Créer un **Point2D** sans paramètre
 - Créer un **Point2D** à partir de deux paramètres
 - Créer un **Point2D** à partir d'un autre Point2D : un constructeur de copie
 - Afficher un **Point2D** en mode textuel de la manière suivante : « **[coordX ; coordy]** » où **coordX** et **coordY** doivent correspondre aux attributs de l'objet
 - « Traduire » un **Point2D** : ajouter un incrément (positif ou négatif) à ses coordonnées
 - Modifier les deux coordonnées d'un **Point2D** à la fois (changer X et Y en une seule méthode)

Point2D
- x: int - y: int
+ Point2D(x: int, y: int) + Point2D(p: Point2D) + Point2D() + setX(x: int) + getX(): int + setY(y: int) + getY(): int + setPosition(x:int, y:int) + translate(dx: int, dy: int) + affiche() + distance (p: Point2D): float



Exercice 3 (2)

- Créez une classe **TestPoint2D.java** dans le même paquetage comportant une fonction principale (« **main** ») :
 - Créez plusieurs objets de type **Point2D** pour illustrer les différents constructeurs que vous avez écrits
 - Illustrez le bon fonctionnement de toutes les méthodes implémentées dans votre classe **Point2D**
- Rendu :
 - Joignez vos classes **Point2D** et **TestPoint2D** à votre rapport
 - Dans le rapport : **Illustrez le bon fonctionnement de vos classes** (captures d'écran)
 - Dans le rapport :
 - Dans l'explorateur de projet, sélectionnez **Point2D.java**
 - Cliquez sur le bouton « **History** » de l'éditeur
 - **Expliquez à quoi sert le bouton History** (illustrez le)
 - **Expliquez pourquoi cela peut être intéressant**

