



3. Orienté Objet

Plan du cours

- Chapitre 1 : Le concept d'Orienté Objet
- Chapitre 2 : Le concept de Classe
- Chapitre 3 : Les méthodes
- Chapitre 4 : Le polymorphisme
- Chapitre 5 : Le constructeur
- Chapitre 6 : Les modificateurs d'accessibilité
- Chapitre 7 : Les accesseurs et mutateurs
- Chapitre 8 : Les types primitifs et les wrappers
- Chapitre 9 : L'autoboxing

Plan du cours

- Chapitre 10 : L'héritage
- Chapitre 11 : Les classes abstraites
- Chapitre 12 : Les interfaces
- Chapitre 13 : Les classes internes, locales et anonymes
- Chapitre 14 : Le destructeur
- Autres informations utiles
 - La classe Random
 - La classe LocalDate
- Comment créer un jar exécutable ?
 - Comment executer le jar généré ?
- Comment créer un installateur de programme pour Windows ?

Le concept d'Orienté Objet

Chapitre 1

Le concept d'Orienté Objet

- Définition :

*« [...] Il consiste en la **définition** et l'**interaction** de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. [...] Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues [...] »*

Programmation orientée objet, [Wikipedia](#)

Le concept d'Orienté Objet

- Concrètement :
 - La Programmation Orientée Objet (P.O.O.) permet de diviser efficacement son code en « classes », qui contiennent chacune des champs et des méthodes spécifiques.
 - L'objectif de la P.O.O. est de diminuer le temps passé à programmer en réutilisant des composants existants.

Le concept d'Orienté Objet

- C'est un concept incontournable en programmation, la plupart des langages actuels utilisent ce concept.
 - Exemples : C#, Python, Ruby, C++, PHP,...

Le concept de classe

Chapitre 2

Le concept de Classe

- Définition :

Une classe est une structure informatique servant de plan, de moule, pour la création d'objets. Elle contient des variables et du code regroupé sous forme de méthodes permettant de définir un comportement.

*Un objet peut être **instancié** à partir d'une classe.*

Le concept de Classe

- Concrètement :
 - Une classe va permettre de définir les caractéristiques spécifiques d'une série d'objets.
 - Une classe va donc servir de moule, grâce auquel seront créé de nouveaux objet
 - Tous ces objets auront les mêmes attributs, mais avec des valeurs différentes

Le concept de Classe

- Exemple : tous les employés possèdent un nom, un prénom, une date de naissance,... Ce sont leurs attributs. On pourrait donc créer une classe Employe qui nous permettrait d'instancier des objets depuis cette classe.

```
public class Employe {  
  
    String nom;  
    String prenom;  
    String dateDeNaissance;  
  
}
```

Le concept de Classe

- Bonnes pratiques :
 - Il est fortement conseillé de ne pas faire de la redondance d'informations entre les attributs. Par exemple, si on a déjà un attribut *dateDeNaissance*, il n'est pas nécessaire d'avoir un attribut *age*, car on peut le calculer.
 - En Java, contrairement à d'autres langages de programmation, il ne peut y avoir qu'une classe par fichier.

Il y a cependant une exception à cette règle : les classes internes (nous reparlerons plus tard)

```
public void demenager(String nouvelleVille) {  
    /*  
     * Bloc de la méthode  
     */  
}
```

Les Méthodes

Chapitre 3

Les Méthodes

- Une méthode est une structure contenant du code.
- Elle est obligatoirement déclarée dans une classe.
- Tous les objets de cette classe implémenteront cette méthode.

Les Méthodes

- Syntaxe
 - Comme nous l'avons déjà vu, une méthode a une syntaxe du type :

```
public void demenager(String nouvelleVille) {  
    /*  
     * Bloc de la méthode  
     */  
}
```

Les Méthodes

- Le modificateur d'accès :
 - Définit qui pourra avoir accès à cette méthode , nous nous intéresserons à ces modificateurs plus tard.
- Le retour d'une méthode
 - Une méthode doit parfois pouvoir renvoyer un résultat, il faut donc spécifier s'il y a oui ou non renvoi d'une valeur, et si oui, de quel type de valeur il s'agit.

Les Méthodes

- Les paramètres
 - Une méthode peut avoir besoin de certaines valeurs pour fonctionner. Il faudra donc les passer en paramètres à l'appel de la méthode.
 - Il est nécessaire de donner le type de paramètres attendus ainsi que de leur donner un nom. Ce nom doit respecter les conventions habituelles des noms de variables.

Les Méthodes

- Une méthode est déterminée par sa signature, c'est-à-dire :
 - Sa classe d'appartenance
 - Son nom
 - Ses paramètres

Les Méthodes

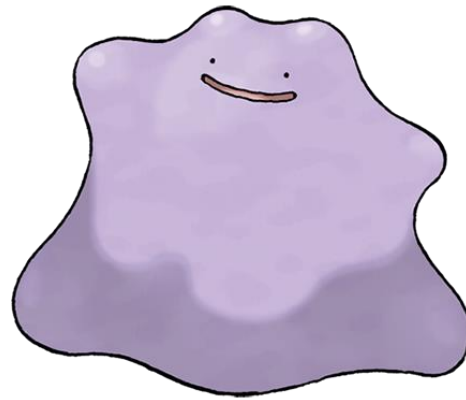
- La surcharge de méthode :
 - Il arrive qu'on ait besoin de « surcharger » une méthode afin d'accepter des paramètres différents :

```
public void demenager(String nouvelleVille) {  
  
    /*  
     * Bloc de la méthode  
     */  
  
}  
  
public void demenager(int nouveauCodePostal) {  
  
    /*  
     * Bloc de la méthode  
     */  
  
}
```

Les Méthodes

- La surcharge de méthode :
 - Fait partie du premier pilier de la programmation orienté objet :

Le polymorphisme



Le Polymorphisme

Chapitre 4

Le Polymorphisme

- On distingue 3 types de polymorphisme :
 1. Le polymorphisme ad hoc
 2. Le polymorphisme paramétrique
 3. Le polymorphisme d'héritage

Le Polymorphisme

- Le polymorphisme ad hoc
 - Permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles
 - Par exemple, la classe FigureComplexe, la classe Image et la classe LienInternet peuvent avoir chacune une fonction "afficher()". Cela permettra de ne pas avoir à se soucier du type de l'objet que l'on a si on souhaite l'afficher à l'écran

[definition-du-polymorphisme](#)

Le Polymorphisme

- Le polymorphisme ad hoc
 - La surcharge d'opérateur
 - Contrairement à des langages tels que C++ et C#, **Java n'autorise pas la surcharge des opérateurs** !
 - Le cas de l'addition de deux String fait partie des spécifications du langage, c'est une exception propre à la classe String. Elle est justifiée par son usage extrêmement courant.

[Spécifications du langage Java](#)

Le Polymorphisme

- Le polymorphisme paramétrique
 - Il permet de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type)
 - Il rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.
 - Ce type de polymorphisme est aussi appelé **généricité**

[definition-du-polymorphisme](#)

Le Polymorphisme

- Le polymorphisme paramétrique

- Exemple :

On peut définir plusieurs méthodes homonymes *addition()* effectuant une somme de valeurs :

- La méthode *int addition(int, int)* pourra retourner la somme de deux entiers
 - La méthode *float addition(float, float)* pourra retourner la somme de deux flottants
 - La méthode *char addition(char, char)* pourra définir au gré de l'auteur la somme de deux caractères

[definition-du-polymorphisme](#)

Le Polymorphisme

- Le polymorphisme d'héritage
 - La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la **spécialisation**.
 - Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque. Ceci permet de faire abstraction des détails des classes spécialisées d'une famille d'objet, en les masquant par une interface commune (qui est la classe de base).

[definition-du-polymorphisme](#)

Le Polymorphisme

- Le polymorphisme d'héritage

- Exemple :

- Imaginons un jeu d'échec comportant des objets *roi*, *reine*, *fou*, *cavalier*, *tour* et *pion*, héritant chacun de l'objet *piece*.
 - La méthode *mouvement()* pourra, grâce au polymorphisme d'héritage, effectuer le mouvement approprié en fonction de la classe de l'objet référencé au moment de l'appel.
 - Cela permettra notamment au programme d'appeler *piece.mouvement()* sans avoir à se préoccuper de la classe de la pièce.

[definition-du-polymorphisme](#)



Le Constructeur

Chapitre 5

Le constructeur

- Le constructeur
 - Un constructeur est une méthode particulière qui permet l'instanciation d'un objet de la classe.
 - Typiquement, le constructeur se charge de réserver de la place mémoire pour tout nouvel objet et d'affecter les valeurs reçues en paramètres aux différents attributs de la classe.

Le constructeur

- Il porte obligatoirement le même nom que celui de la classe.
- Il ne retourne jamais de valeur, il n'y a donc pas d'indication de type de retour (même pas void!).

Le constructeur

- Syntaxe :
 - Dans la classe

```
public Personne(String nomArg, String prenomArg, int ageArg,  
    String adresseArg, String codePostalArg, String villeArg) {  
    nom = nomArg;  
    prenom = prenomArg;  
    age = ageArg;  
    adresse = adresseArg;  
    codePostal = codePostalArg;  
    ville = villeArg;  
}
```

```
Personne p = new Personne("Doe", "Jane", 99, "Rue de Jane", "5000", "Namur");
```


Le mot clé « this »

- **this** : désigne, à l'intérieur d'une classe, l'instance courante de la classe elle-même.
- On l'utilise souvent dans le constructeur afin de différencier un attribut de la classe et un paramètre qui portent le même nom :

```
public Personne(String nom, String prenom, int age, String adresse, String codePostal, String ville) {  
    this.nom = nom;  
    this.prenom = prenom;  
    this.age = age;  
    this.adresse = adresse;  
    this.codePostal = codePostal;  
    this.ville = ville;  
}
```

Le constructeur

- Notes :
 - En l'absence de définition d'un constructeur, la classe contient (sans qu'on puisse le voir) un constructeur par défaut.
 - Il est tout à fait possible de définir plusieurs constructeurs pour une seule classe. Selon les paramètres qui envoyés, le compilateur sélectionne le constructeur correspondant.

Les Modificateurs d'Accessibilité

Chapitre 6

Les modificateurs d'accessibilité

- Afin d'éviter toute modification indésirable des données de notre programme, on utilise des modificateurs d'accessibilité.
- Ils nous permettent de choisir qui pourra utiliser nos méthodes, nos classes et nos attributs.

Les modificateurs d'accessibilité

- Signification :
 - public : toutes les classes peuvent y accéder
 - protected : seules les classes dérivées et les classes du même package peuvent y accéder
 - private : seulement accessible depuis l'intérieur de la classe où il est défini
 - (aucun) : par défaut, seules les classes du même package peuvent y accéder

Les modificateurs d'accessibilité

- En résumé :

	private	Aucun	protected	public
Accès depuis la classe	Oui	Oui	Oui	Oui
Accès depuis une classe du même package	Non	Oui	Oui	Oui
Accès depuis une sous-classe	Non	Non	Oui	Oui
Accès depuis toute autre classe	Non	Non	Non	Oui

Autres modificateurs

- Signification :
 - abstract :
 - indique une classe ou une méthode est abstraite (les méthodes ne contiennent pas de corps de méthode)
 - final :
 - devant un attribut, déclare une constante
 - devant une méthode, déclare une méthode ne pouvant pas être redéfinie dans une classe dérivée
 - devant une classe, déclare une classe ne pouvant pas être dérivée
 - static :
 - devant un attribut, déclare qu'il est partagé par toutes les instances de sa classe
 - Devant une méthode, déclare qu'elle peut être appelée sans instancier sa classe

Les Accesseurs et mutateurs

Chapitre 7

Accesseurs et mutateurs

- En Java les accesseurs et mutateurs sont de simples méthodes.
- Ces méthodes permettent de continuer à accéder aux attributs de nos objets, tout en les protégeant de modifications non-désirée.
- L'utilisation des accesseurs et des mutateurs est le deuxième pilier de la programmation orientée objet :

L'encapsulation



Accesseurs et mutateurs

- L'encapsulation
 - L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet
 - C'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés
 - L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet

[Encapsulation des données](#)

Accesseurs et mutateurs

- Le masquage des informations
 - L'utilisateur d'une classe n'a pas besoin de savoir de quelle façon sont structurées les données dans l'objet
 - Ainsi, en interdisant à l'utilisateur de modifier directement les attributs, et en l'obligeant à utiliser les fonctions définies pour cet usage (appelées **interfaces**), on est capable de s'assurer de l'intégrité des données
 - On pourra par exemple s'assurer que le type des données fournies est conforme à nos attentes, ou encore que les données se trouvent bien dans l'intervalle attendu

[Encapsulation des données](#)

Accesseurs et mutateurs

- Fonctionnement :
 - Dans notre classe, passer tous les attributs en private.
 - Créer deux nouvelles méthodes par attribut :
 - Un getter et un setter :

```
public String getNom() {  
    return nom;  
}  
  
public void setNom(String nom) {  
    this.nom = nom;  
}
```

Types primitifs, types objets et types Wrappers

Chapitre 8

Types primitifs

- Comme nous l'avons vu, tous les types de variables, à l'exception de String, sont des types primitifs. Cela signifie qu'ils bénéficient d'une dérogation spéciale en Java : ce ne sont pas des classes !

Types objets

- String, par contre, est une classe et permet de créer des objets String.
- Vu que c'est un objet, il possède des méthodes propres, comme par exemple la méthode *equals*.

Types Wrappers

- Les « Wrappers », ou enveloppeurs en français, permettent de manipuler les types primitifs à l'aide de méthodes incluses spécialement pour ces wrappers.
- Chaque type primitif possède un type wrapper associé.

Types Wrappers

Type Wrapper	Type Primitif
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean

Le Wrapper utilise plus d'espace mémoire que le type primitif, un int prend 4 octets en mémoire, tandis qu'un Integer en prendra 32 !

De plus, un Wrapper ne peut pas être modifié, il faut créer un nouvel objet et détruire l'ancien à chaque changement de valeur...

Autoboxing

Chapitre 9

Autoboxing

- Boxing : Transformation d'un type primitif dans un type objet
- Unboxing : Transformation d'un type objet en type primitif

Autoboxing

- L'autoboxing, comme son nom l'indique, se fait depuis Java 5 de façon implicite et automatique par le compilateur

```
int prim = 8;  
Integer wrapp = prim;  
int reprim = wrapp;
```

Autoboxing

- L'autoboxing est nécessaire pour utiliser des ArrayList typées, en effet, comme nous l'avons vu, les types primitifs ne sont pas des objets!



L'héritage

Chapitre 10

L'héritage

- Troisième pilier de la programmation orientée objet
- Concept d'héritage :
 - Une classe peut hériter d'une autre classe. Cela permet de créer une hiérarchie logique de classes.
 - On indique d'une classe hérite d'une autre grâce au mot-clé **extends** :

```
public class Employe extends Personne {
```

L'héritage

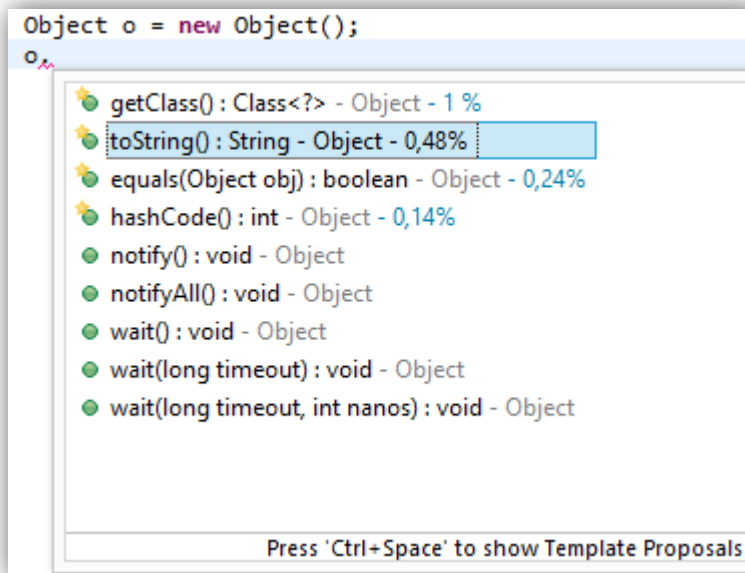
- Exemple d'héritage :
 - Dans une entreprise travaillent des cadres et des employés. À quoi ressembleraient les classes Cadre et Employe si on n'utilise pas le concept d'héritage? Et si on l'utilise?

L'héritage

- Une classe qui hérite d'une autre classe est appelée sous-classe, classe enfant, classe fille ou encore classe dérivée.
- Une classe dont hérite(nt) une ou plusieurs classes est appelée superclasse ou classe mère.
- En Java, dans le cas où une classe n'hérite explicitement d'aucune autre classe, elle hérite en fait implicitement de la classe Object.

L'héritage

- La classe Object :
 - Est la superclasse par défaut de toutes les classes
 - Définit déjà certaines méthodes de base :



L'héritage

- `getClass()` : renvoie la classe de l'objet
- `toString()` : renvoie la représentation de l'objet sous la forme d'une chaîne de caractères
- `equals(Object obj)` : renvoie `true` si les deux objets sont les mêmes, `false` si ils sont différents

Créez une classe `Personne`, héritant implicitement de la classe `Object`.

Que se passe-t-il si vous tentez d'exécuter ce code :

```
Personne personnel = new Personne("Gates", "Bill", 59, "Rue de Microsoft", "5000", "Silicon Valley");  
System.out.println(personnel);
```

Souvent, on choisit donc de redéfinir les méthodes `toString()` et `equals(Object obj)` des classes que l'on crée, afin de faciliter leur manipulation.

L'héritage

- Les méthodes et classes "final"
 - Nous avons déjà évoqué le fait qu'une classe peut être qualifiée de "final" :
 - Elle ne peut être dérivée.
 - Il est également possible de désigner une méthode comme "final" :
 - *elle ne pourra plus être redéfinie* dans les sous-classes éventuelles.

L'héritage

- Les méthodes et classes "final"
 - Les raisons d'utiliser des méthodes interdites à la redéfinition sont de deux ordres :
 - La sécurité
 - L'optimisation

L'héritage

- Les méthodes et classes "final"
 - La sécurité
 - Comme pour les classes, on assure ainsi que la méthode ne pourra être détournée du but pour lequel elle a été créée
 - L'exemple classique est celui d'une méthode de validation de mot de passe.

L'héritage

- Les méthodes et classes "final"
 - L'optimisation
 - Pour une méthode normale d'un objet d'une hiérarchie, un appel doit être résolu en recherchant le type effectif de l'objet appelant afin d'utiliser la bonne version de la méthode : Ceci prend du temps
 - Dans le cas d'une méthode final, cette recherche est inutile et on peut même remplacer l'appel de la méthode par son code; on retrouve ici le concept de méthode inline du C++

Les Classes abstraites

Chapitre 11

Les Classes Abstraites

- Définition :

« Une classe abstraite est une classe qui ne peut être instanciée. Elle permet de faire de la factorisation de code, en ne donnant qu'une implémentation partielle. »

Les Classes Abstraites

- Utilité :
 - Permet d'être la superclasse de plusieurs classes dérivées sans répéter des attributs ou des méthodes communes.
 - Permet de contenir des méthodes abstraites, c'est-à-dire des méthodes qui n'ont pas de corps. Ces méthodes devront donc être définies par les classes enfants.

Les Classes Abstraites

- Syntaxe et exemple :
 - Classe abstraite :

```
public abstract class Compte {  
  
    private int solde;  
  
    public int getSolde() {  
        return solde;  
    }  
  
    public abstract void retrait(int montant);  
  
    public abstract void depot(int montant);  
}
```

Les Classes Abstraites

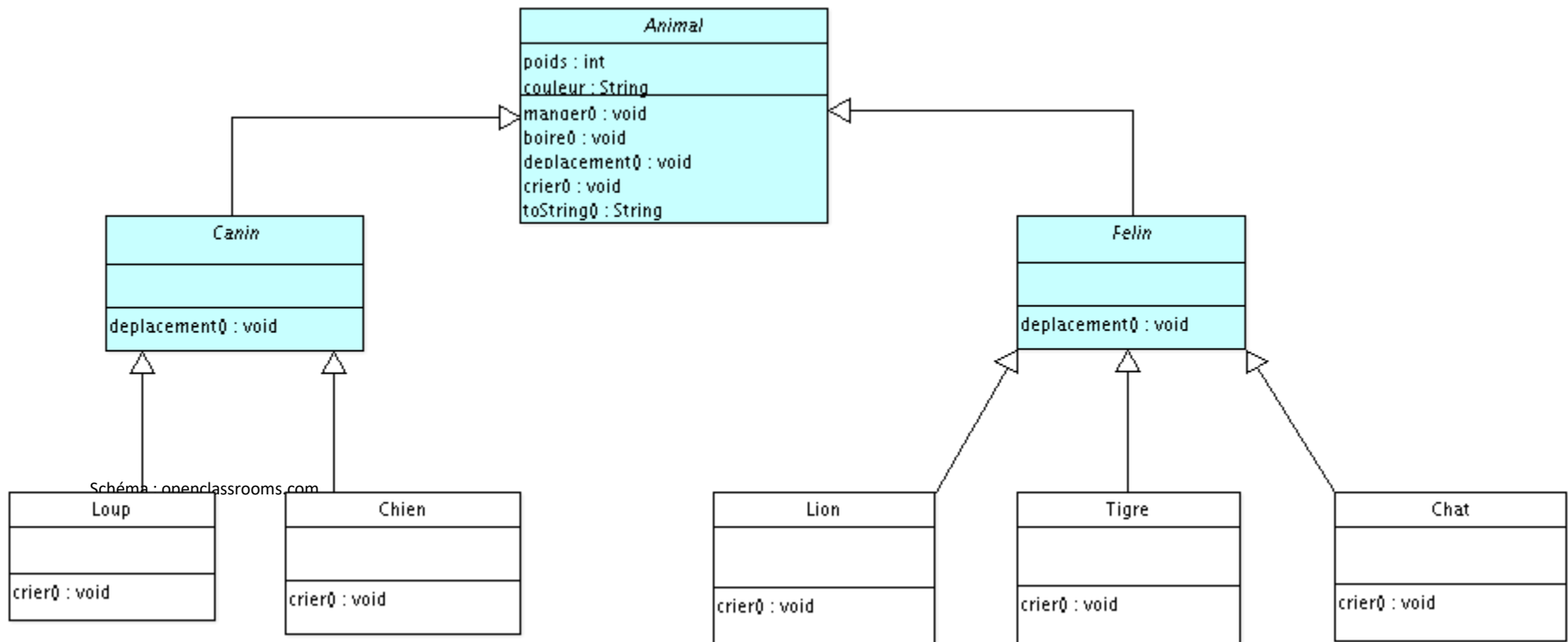
- Classes enfants :

```
public class CompteEpargne extends Compte {  
  
    @Override  
    public void retrait(int montant) {  
        // DEFINITION DE LA METHODE  
    }  
  
    @Override  
    public void depot(int montant) {  
        // DEFINITION DE LA METHODE  
    }  
}
```

```
public class CompteBancaire extends Compte {  
  
    @Override  
    public void depot(int montant) {  
        // METHODE DEPOT  
    }  
  
    @Override  
    public void retrait(int montant) {  
        // METHODE RETRAIT  
    }  
  
    public void modifierTitulaire(Personne nouveauTitulaire) {  
        // METHODE MODIFIER TITULAIRE  
    }  
}
```

Les Classes Abstraites

- Exemple d'utilité :



Les Interfaces

Chapitre 12

Les Interfaces

- Définition :

« Une interface fonctionne comme une classe abstraite dont toutes les méthodes sont abstraites, et dont tous les attributs sont constants. Une interface définit le comportement de classes qui l'implémentent, sans implémenter elle-même ce comportement. »

Les Interfaces

- Une interface contient une liste de signatures de méthodes.
- C'est un « contrat » qu'on peut demander à une classe de respecter.
- Ce « contrat » s'établit lorsqu'une classe implémente cette interface.

Les Interfaces

- Une classe qui implémente une interface doit *au minimum* donner une implémentation pour chacune des méthodes que celle-ci contient.

Les Interfaces

- Les interfaces ont trois grandes utilités :
 - Agir comme un contrat, c'est-à-dire obliger les classes qui l'implémentent à définir toutes ses méthodes
 - Simuler l'héritage multiple
 - Restreindre l'accès aux méthodes de la classe, lorsqu'on utilise l'interface

Les Interfaces

- Notes :
 - Une interface n'a jamais de constructeur, elle ne permet pas de construire directement des objets.
 - Une interface ne peut contenir que des champs statiques.

Les Interfaces

- Syntaxe et exemple :

```
public interface VehiculeInterface {  
    public void rouler();  
    public void freiner();  
}
```

Les Interfaces

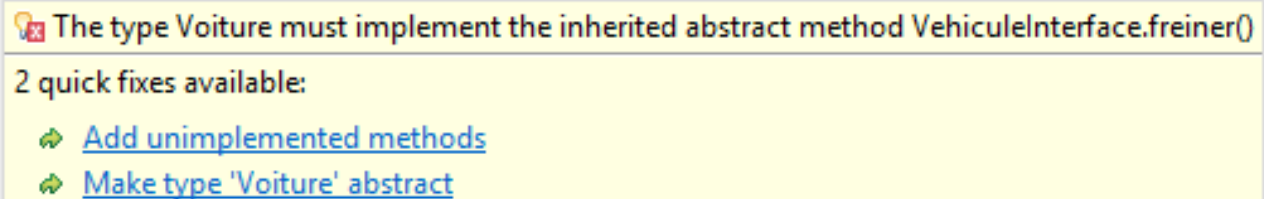
- Exemple de contrat :

- Interface :

```
public interface VehiculeInterface {  
  
    public void rouler();  
  
    public void freiner();  
  
}
```

- Classe :

```
public class Voiture implements VehiculeInterface {  
  
}
```



The type Voiture must implement the inherited abstract method VehiculeInterface.freiner()

2 quick fixes available:

- [Add unimplemented methods](#)
- [Make type 'Voiture' abstract](#)

Les Interfaces

- Exemple d'héritage multiple

- Interfaces :

```
public interface VehiculeInterface {  
    public void rouler();  
    public void freiner();  
}
```

```
public interface BateauInterface {  
    public void naviguer();  
}
```

- Classe :

```
public class VehiculeAmphibie implements VehiculeInterface, BateauInterface {  
    @Override  
    public void naviguer() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void rouler() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void freiner() {  
        // TODO Auto-generated method stub  
    }  
}
```

Les Interfaces

- Exemple de restriction :
 - Interface :

```
public interface ClientCompte {  
    public int getSolde();  
    public void depot(int montant);  
    public void retrait(int montant);  
}
```

Les Interfaces

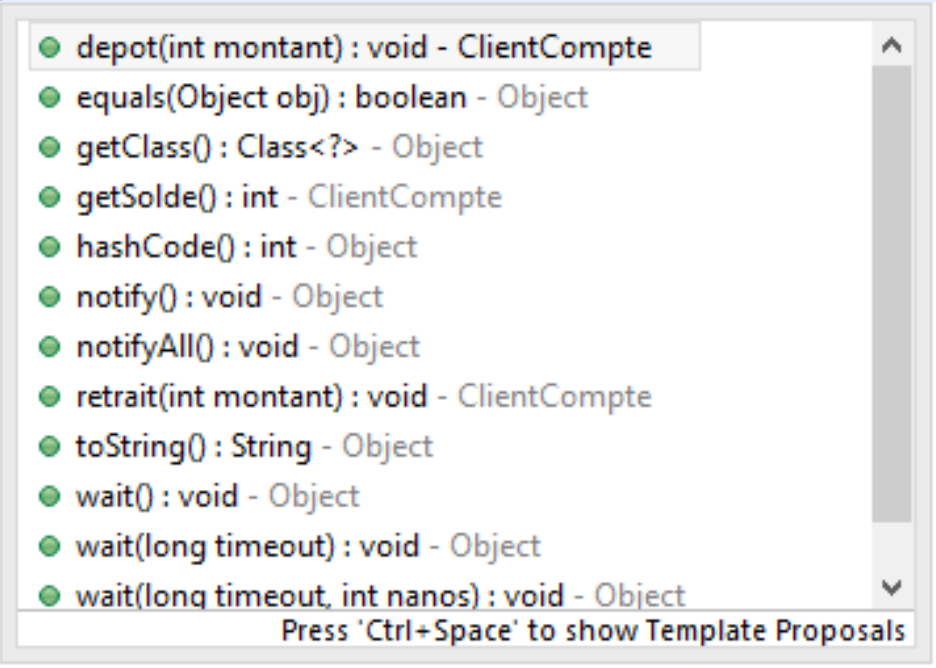
- Classe :

```
public class CompteBancaire implements ClientCompte {  
  
    @Override  
    public int getSolde() {  
        // METHODE GET SOLDE  
        return 0;  
    }  
  
    @Override  
    public void depot(int montant) {  
        // METHODE DEPOT  
    }  
  
    @Override  
    public void retrait(int montant) {  
        // METHODE RETRAIT  
    }  
  
    public void modifierTitulaire(Personne nouveauTitulaire) {  
        // METHODE MODIFIER TITULAIRE  
    }  
}
```


Les Interfaces

- Main :

```
ClientCompte monCompte = new CompteBancaire();  
monCompte
```



Press 'Ctrl+Space' to show Template Proposals

Le client n'a donc pas accès à la méthode *modifierTitulaire*!

Les classes internes, locales et anonymes

Chapitre 13

Les classes internes

- Une classe interne (« Inner class ») est une classe à l'intérieur d'une autre classe.

Les classes internes

- Elle peut être statique
 - Dans ce cas elle est interne seulement pour des questions de logique.

Les classes internes

- Si elle n'est pas statique
 - Il faut y accéder depuis la classe qui la contient. On doit donc obligatoirement instancier sa classe englobante.
 - Elle a d'office un accès total aux attributs de la classe englobante.

Les classes internes

- Utilité
 - Quand deux classes sont très liées, indissociables.
 - Quand la classe interne n'a pas de raison d'exister sans sa classe englobante.
- Exemple :
 - On crée un jeu d'échecs, avec une classe « Échiquier ». Celle-ci contient une classe interne « Case ».

Les classes locales

- Une classe locale est une classe interne définie dans un bloc. Sa portée est donc limitée à ce bloc. Elle ne peut pas être statique.
- Exemple :
 - Une méthode peut créer une classe dont elle a besoin pour exécuter un calcul, et retourner simplement le résultat.

Les classes anonymes

- Une classe anonyme est une classe interne à laquelle on ne donne pas de nom.
 - Très utilisées pour implémenter les méthodes d'un Listener
 - Permet de redéfinir des méthodes d'une superclasse, ou d'implémenter une interface.

```
Compteur counterAno = new Compteur() {  
  
    @Override  
    public void sayClic() {  
        System.out.println("Clac!");  
    }  
  
};
```




Le Destructeur

Chapitre 14

Le destructeur

- En Java il n'est pas possible de forcer la libération d'espace mémoire
- La libération d'espace mémoire est réalisée de manière automatique par le Garbage Collector (ramasse-miette) à son rythme
- Il est toutefois possible de préciser dans une méthode appelée **finalize()** les opérations à effectuer lors de la libération

Le destructeur

- `finalize()`

```
protected void finalize() throws Throwable {  
    try {  
        ...  
    } finally {  
        super.finalize();  
    }  
}
```

- Cette méthode peut, par exemple, fermer un fichier, clôturer une session database, ...
- Il est important que cette méthode invoque, d'une manière ou d'une autre, la méthode *finalize()* de sa superclasse, afin d'effectuer le travail correspondant.

Autres Informations Utiles

La classe Random

- Cette classe permet de générer un nombre aléatoire.
- Exemple : `import java.util.Random;`

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Random rand = new Random();  
  
        for (int i = 0; i < 100; i++) {  
            int randomInt = rand.nextInt(10);  
  
            System.out.println(randomInt);  
        }  
    }  
}
```

La classe LocalDate

- Cette classe remplace depuis Java 8 la classe Date, qui souffrait de nombreux manques et d'incohérences.

1.

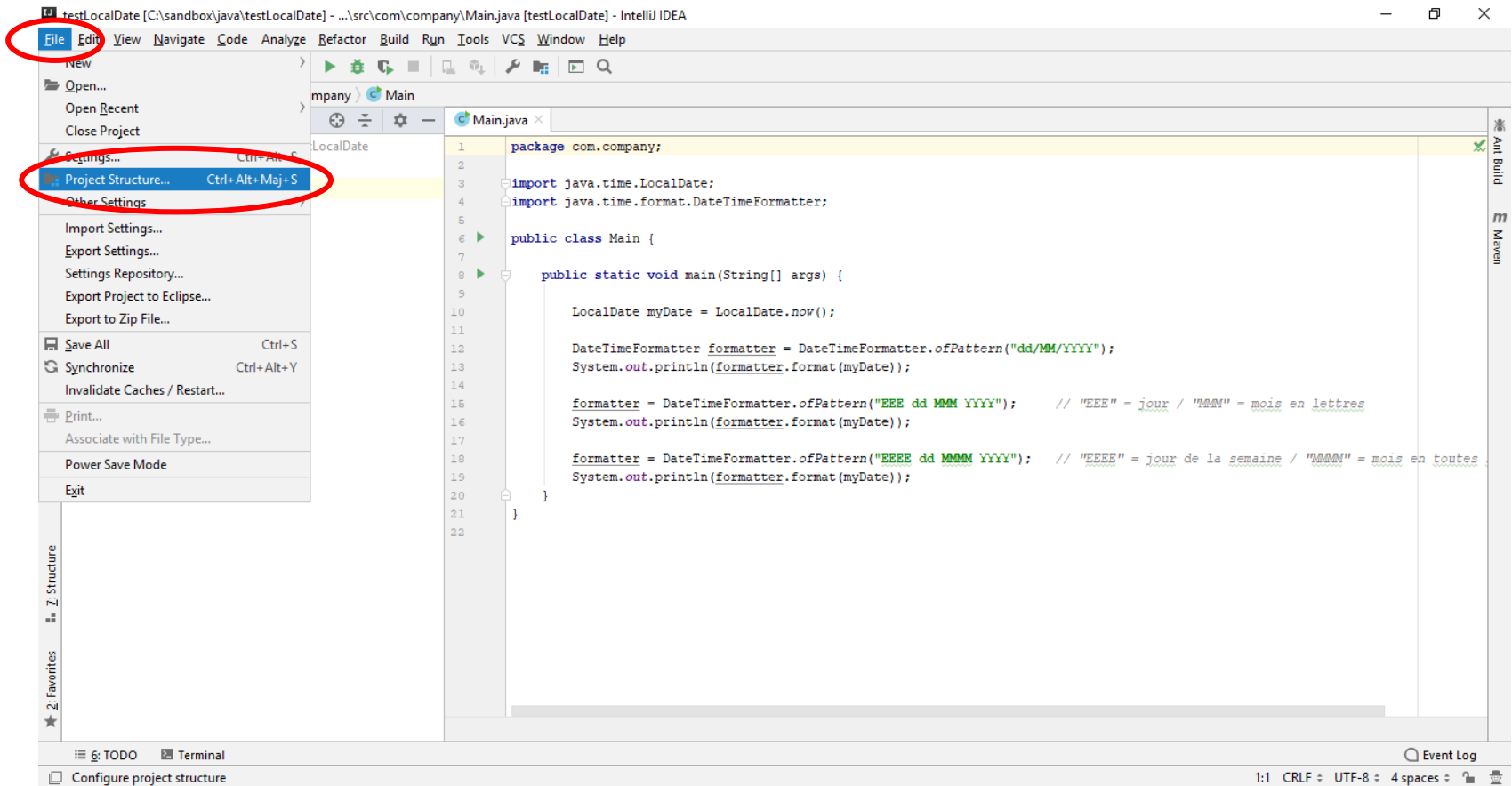
```
LocalDate today = LocalDate.now();  
LocalDate tomorrow = today.plus(1, ChronoUnit.DAYS);  
LocalDate yesterday = tomorrow.minusDays(2);
```
2.

```
LocalDate myDate = LocalDate.now();  
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/YYYY");  
System.out.println(formatter.format(myDate));  
  
formatter = DateTimeFormatter.ofPattern("EEEE dd MMMM YYYY")  
System.out.println(formatter.format(myDate));
```

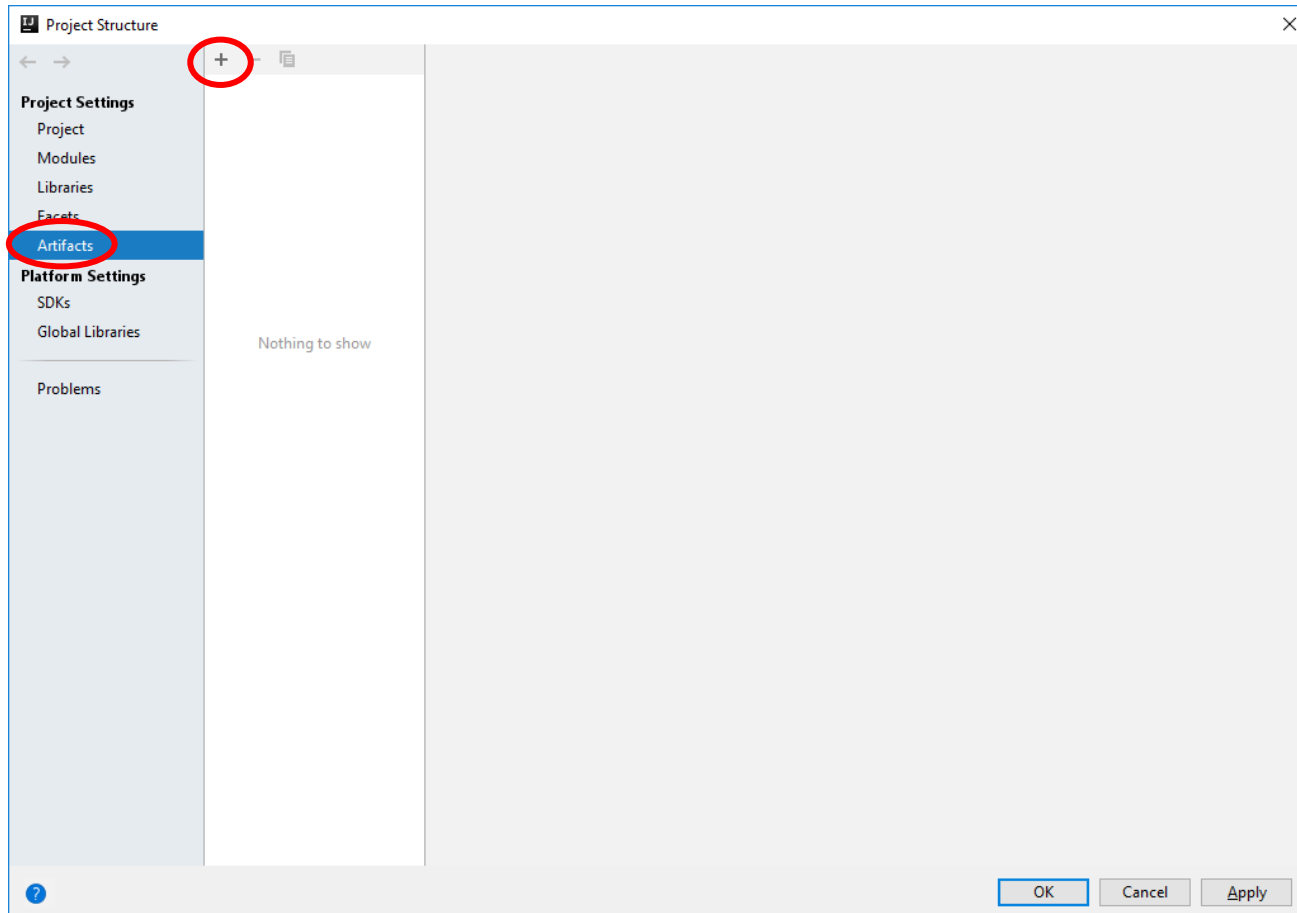
- Il existe également les nouvelles classes LocalTime et LocalDateTime.

Comment créer un jar exécutable ?

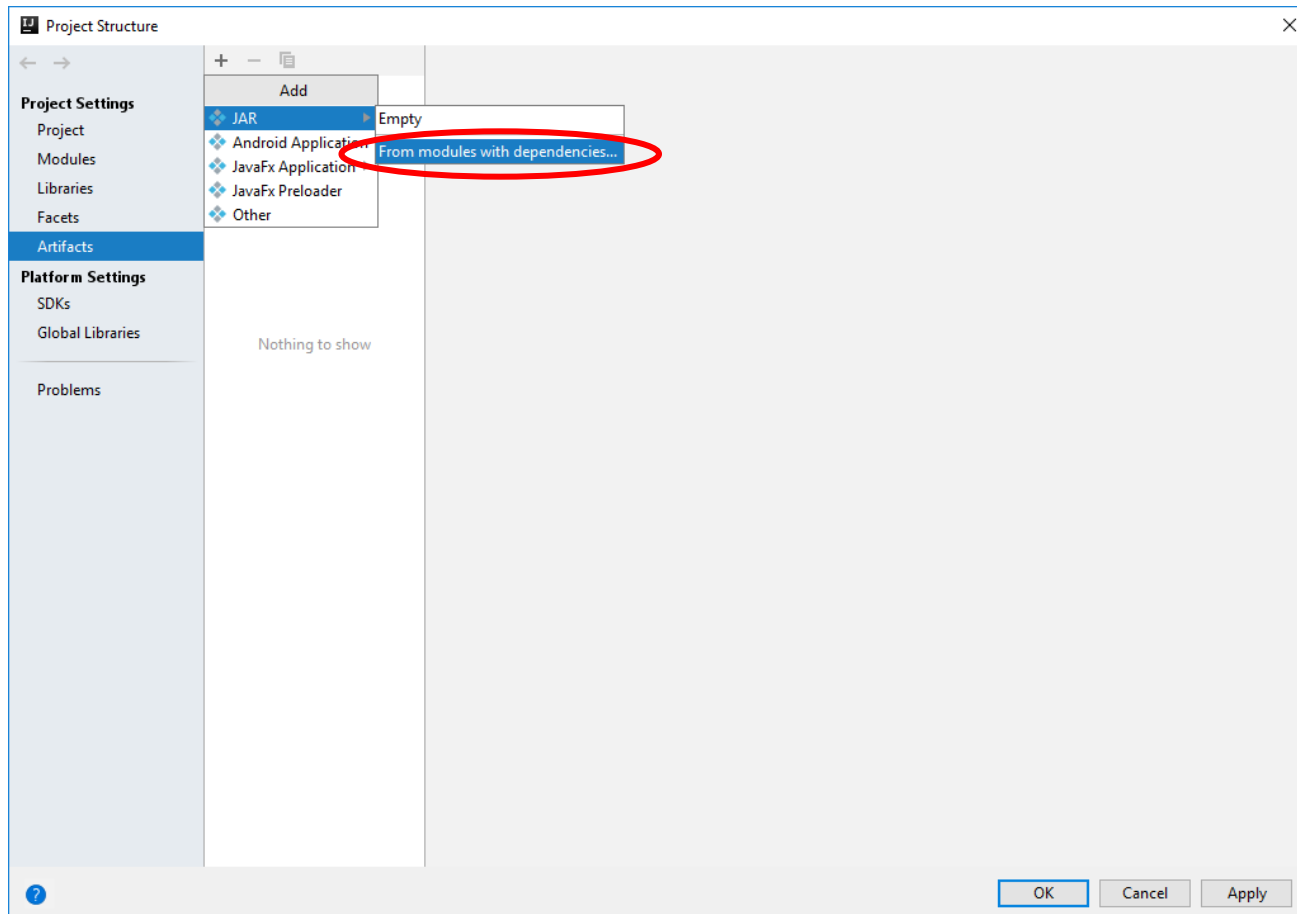
Comment créer un jar exécutable ?



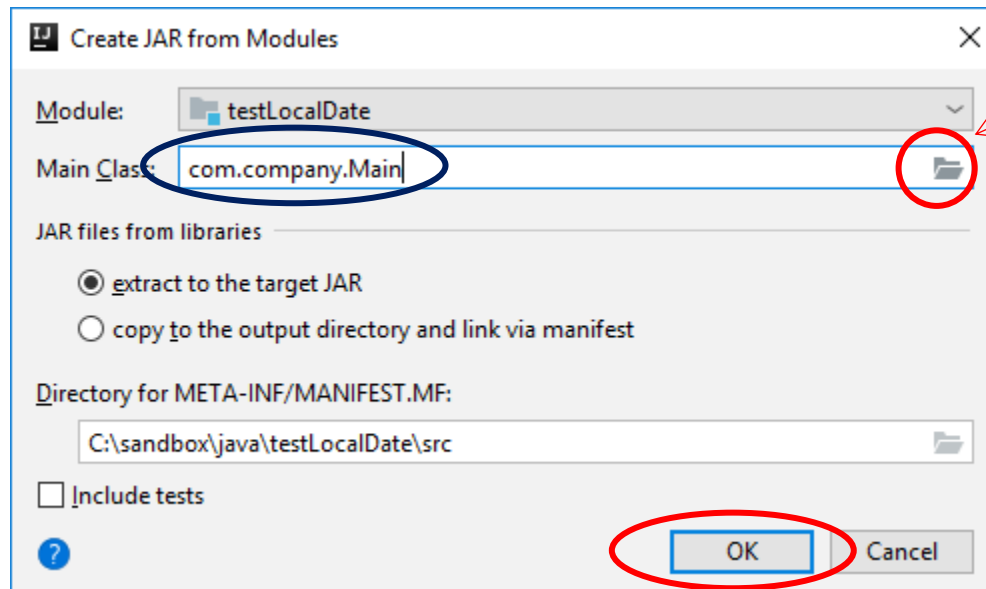
Comment créer un jar exécutable ?



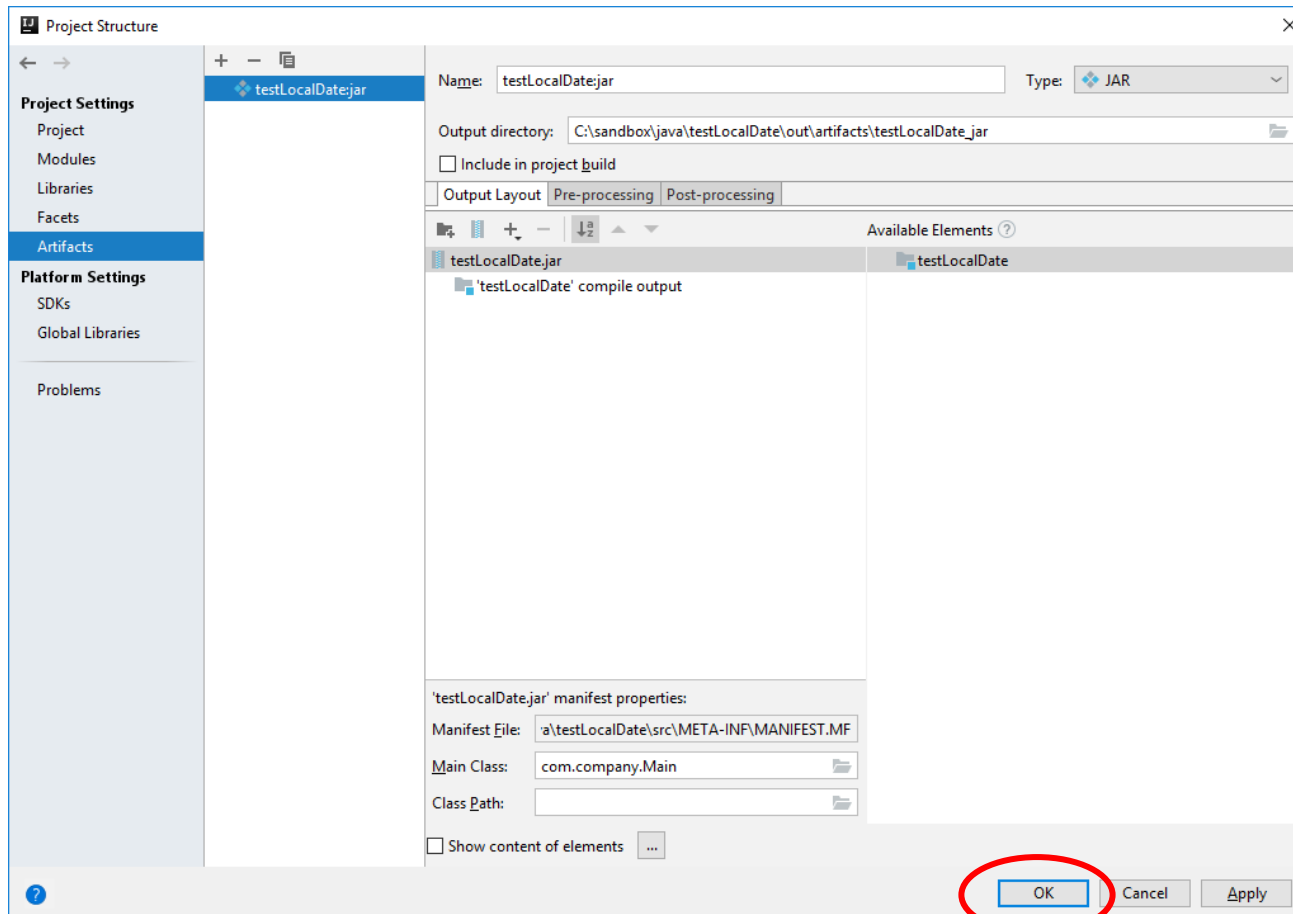
Comment créer un jar exécutable ?



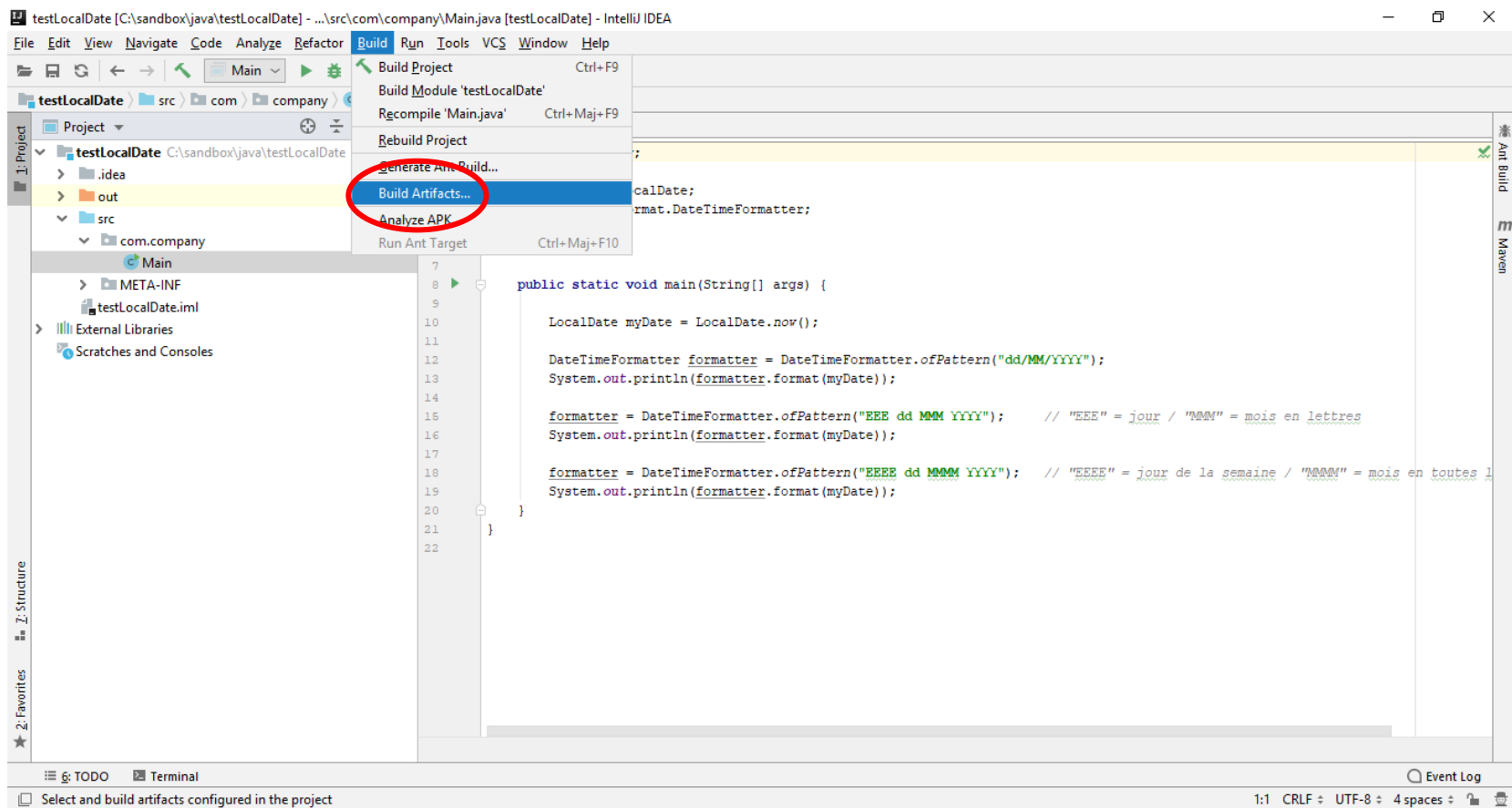
Comment créer un jar exécutable ?



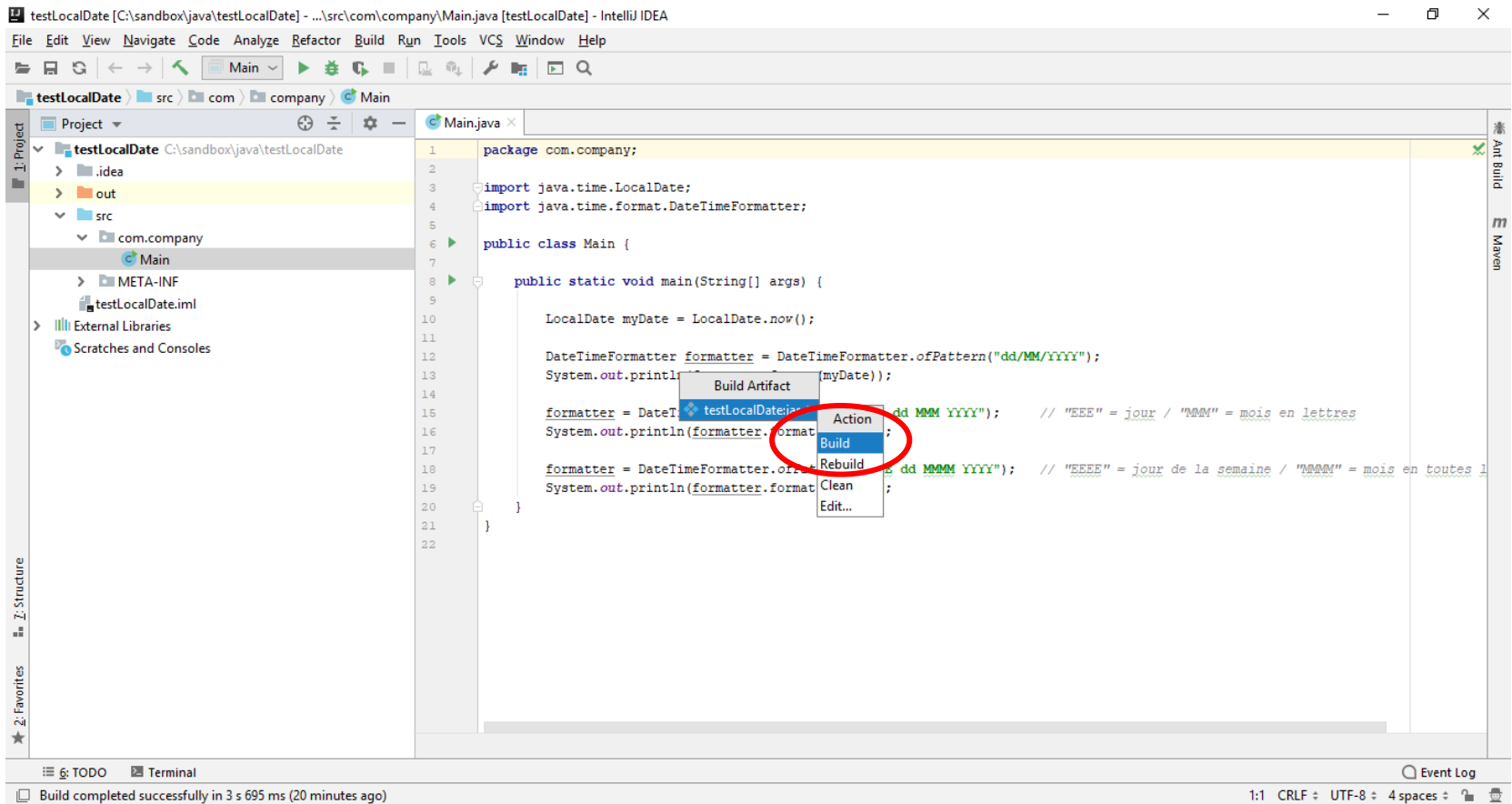
Comment créer un jar exécutable ?



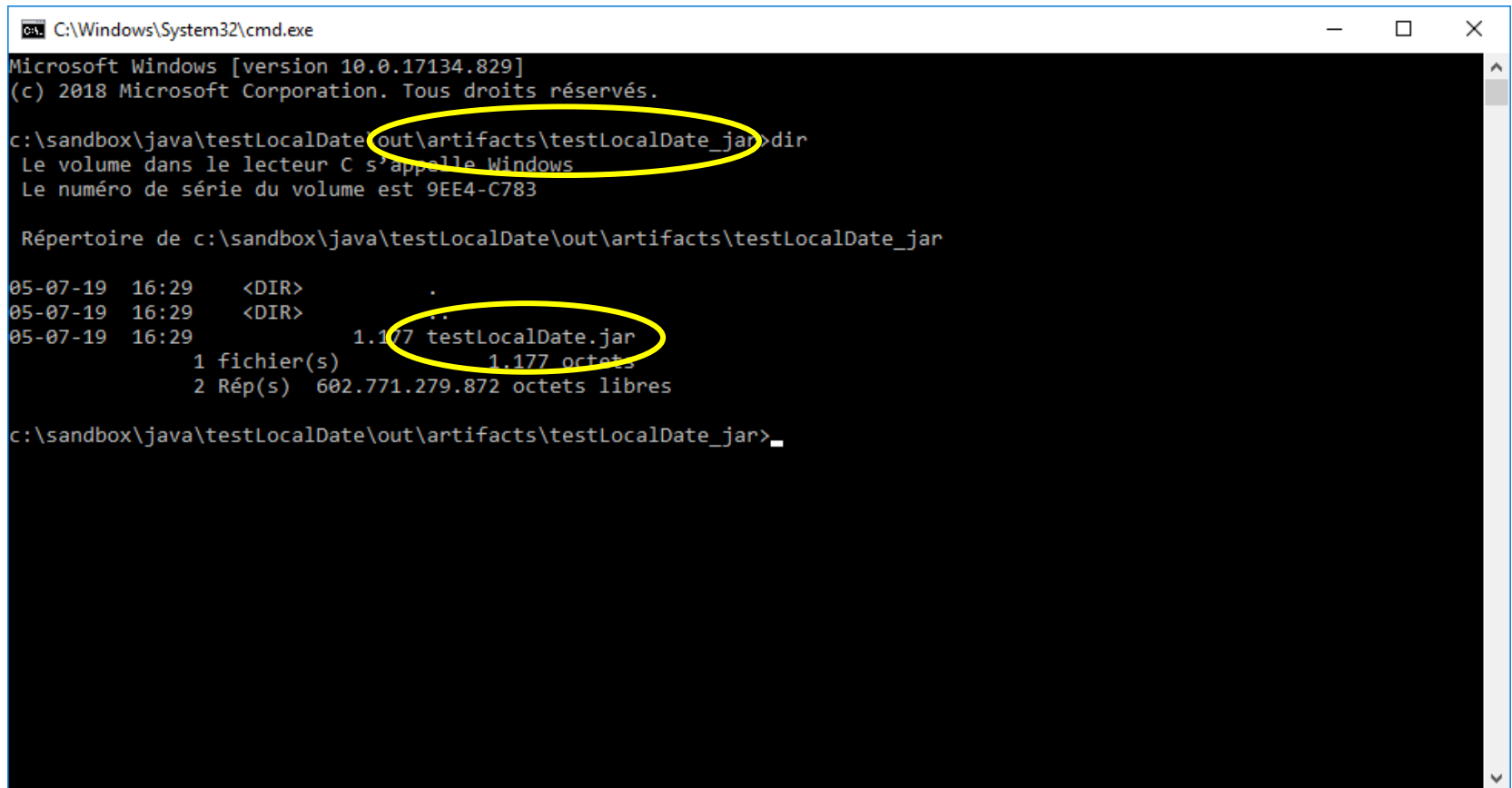
Comment créer un jar exécutable ?



Comment créer un jar exécutable ?



Comment executer le jar généré ?



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.17134.829]
(c) 2018 Microsoft Corporation. Tous droits réservés.

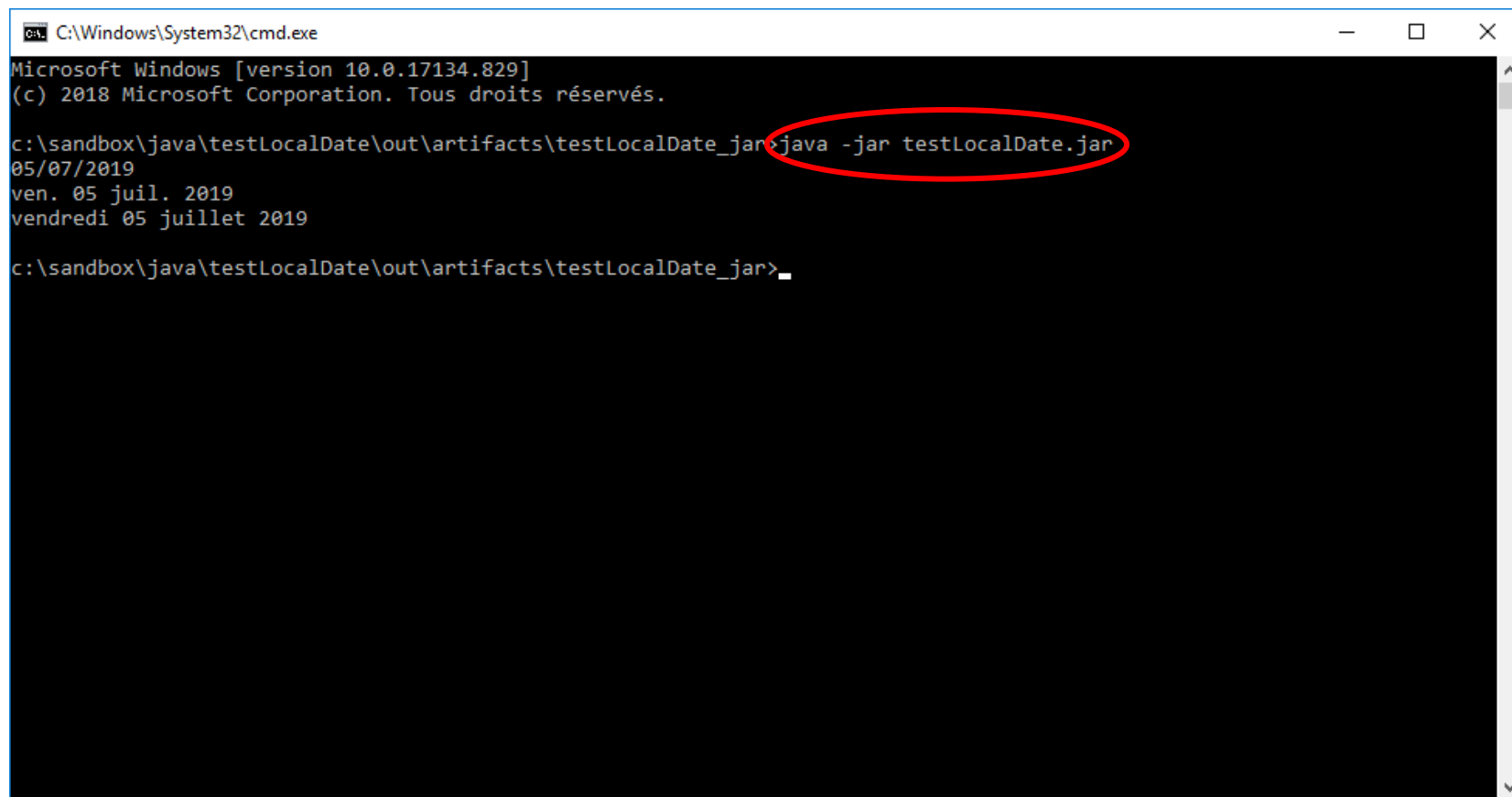
c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>dir
Le volume dans le lecteur C s'appelle Windows
Le numéro de série du volume est 9EE4-C783

Répertoire de c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar

05-07-19  16:29    <DIR>          .
05-07-19  16:29    <DIR>          ..
05-07-19  16:29         1.177 testLocalDate.jar
               1 fichier(s)             1.177 octets
               2 Rép(s)  602.771.279.872 octets libres

c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>
```

Comment executer le jar généré ?



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.17134.829]
(c) 2018 Microsoft Corporation. Tous droits réservés.

c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>java -jar testLocalDate.jar
05/07/2019
ven. 05 juil. 2019
vendredi 05 juillet 2019

c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>
```

The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe". The window contains the following text: "Microsoft Windows [version 10.0.17134.829]", "(c) 2018 Microsoft Corporation. Tous droits réservés.", "c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>java -jar testLocalDate.jar", "05/07/2019", "ven. 05 juil. 2019", "vendredi 05 juillet 2019", and "c:\sandbox\java\testLocalDate\out\artifacts\testLocalDate_jar>". The command "java -jar testLocalDate.jar" is circled in red.

Comment créer un installateur de programme pour windows?

Comment créer un installateur de programme pour Windows ?

- Passage par des programmes tiers :
 - InstallShield
 - Très connu
 - Très largement utilisé
 - Payant
 - NullSoft Install System (NSIS)
 - Gratuit
 - InnoSetup
 - Facile à utiliser
 - Personnalisable
 - Multilingue
 - Open Source
 - Gratuit

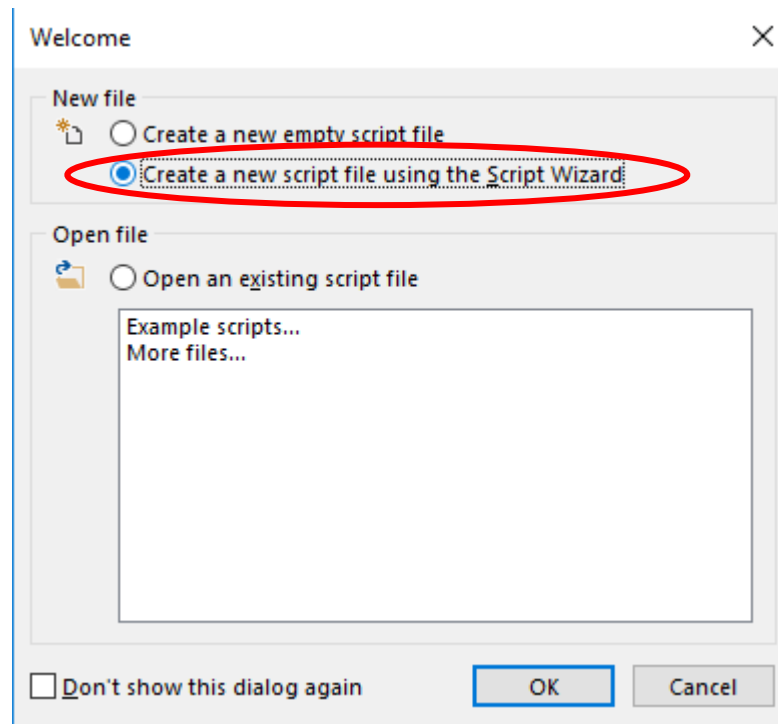
[creer-installateur-programme](#)

Comment créer un installateur de programme pour Windows ?

- InnoSetup
 - Installation
 - <http://www.jrsoftware.org/isinfo.php>

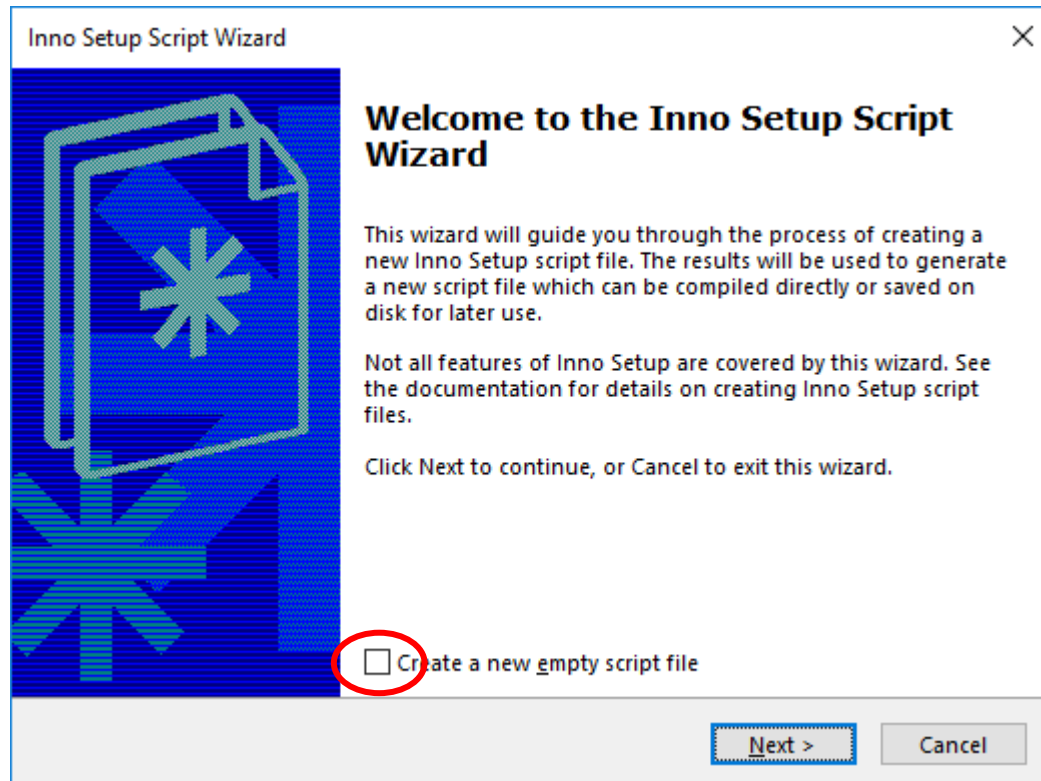
Comment créer un installateur de programme pour Windows ?

- InnoSetup



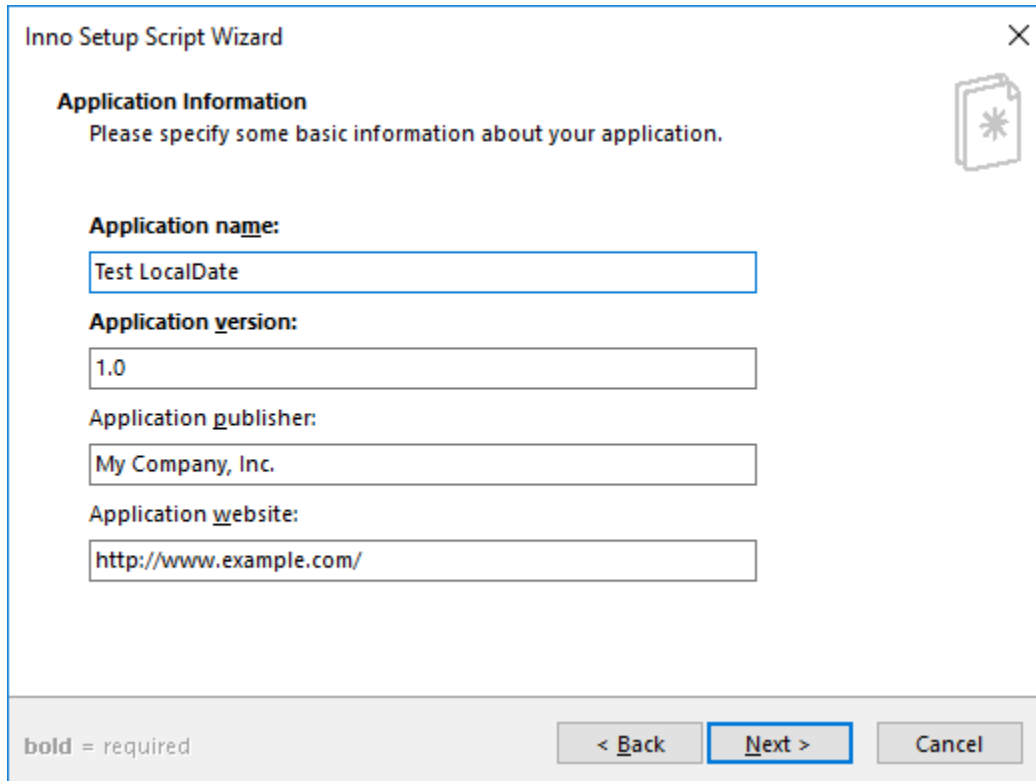
Comment créer un installateur de programme pour Windows ?

- InnoSetup
 - **NE PAS COCHER !**



Comment créer un installateur de programme pour Windows ?

- InnoSetup



The screenshot shows the 'Inno Setup Script Wizard' dialog box. The title bar reads 'Inno Setup Script Wizard' with a close button (X) on the right. The main heading is 'Application Information' with a sub-instruction: 'Please specify some basic information about your application.' To the right of the text is a small icon of a document with a star. Below the instruction are four text input fields, each preceded by a label: 'Application name:' with the value 'Test LocalDate', 'Application version:' with the value '1.0', 'Application publisher:' with the value 'My Company, Inc.', and 'Application website:' with the value 'http://www.example.com/'. At the bottom left, a legend states 'bold = required'. At the bottom right are three buttons: '< Back', 'Next >' (which is highlighted with a blue border), and 'Cancel'.

Inno Setup Script Wizard

Application Information
Please specify some basic information about your application.

Application name:
Test LocalDate

Application version:
1.0

Application publisher:
My Company, Inc.

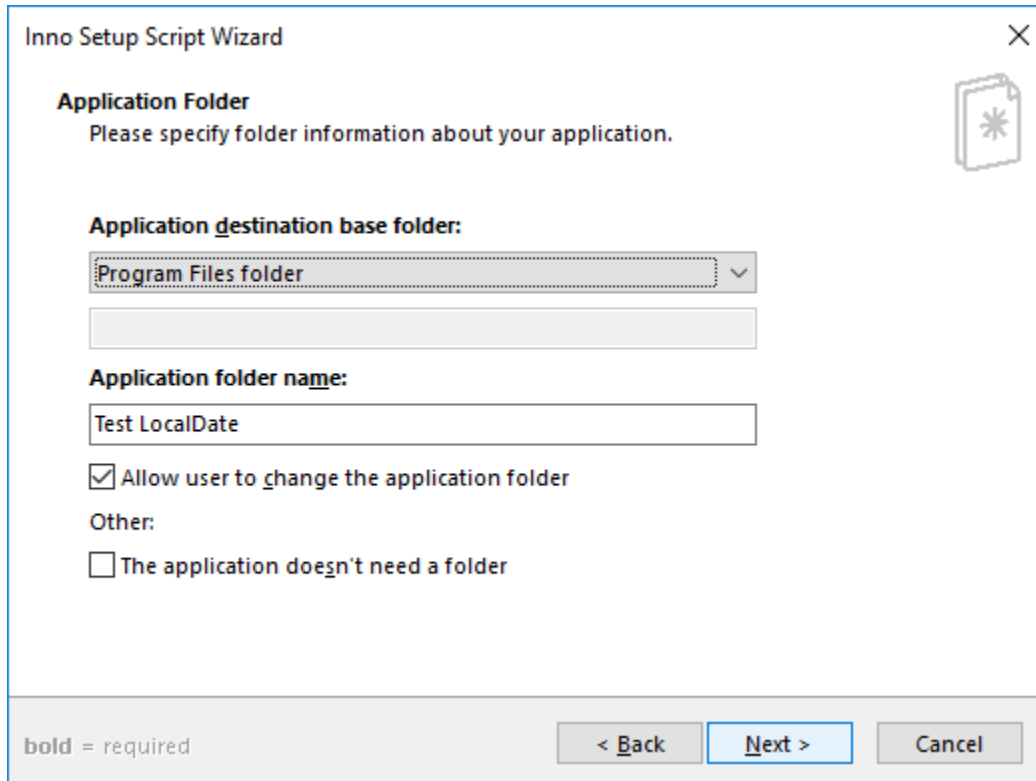
Application website:
http://www.example.com/

bold = required

< Back Next > Cancel

Comment créer un installateur de programme pour Windows ?

- InnoSetup



The screenshot shows the 'Inno Setup Script Wizard' window, specifically the 'Application Folder' step. The window has a title bar with a close button (X) and a help icon (question mark). The main text reads 'Application Folder' and 'Please specify folder information about your application.' Below this, there are three input fields: 'Application destination base folder:' with a dropdown menu showing 'Program Files folder', an empty text box, and 'Application folder name:' with a text box containing 'Test LocalDate'. There are two checkboxes: 'Allow user to change the application folder' (checked) and 'The application doesn't need a folder' (unchecked). At the bottom, there is a legend 'bold = required' and three buttons: '< Back', 'Next >' (highlighted), and 'Cancel'.

Inno Setup Script Wizard

Application Folder
Please specify folder information about your application.

Application destination base folder:

Program Files folder

Application folder name:

Test LocalDate

☒ Allow user to change the application folder

Other:

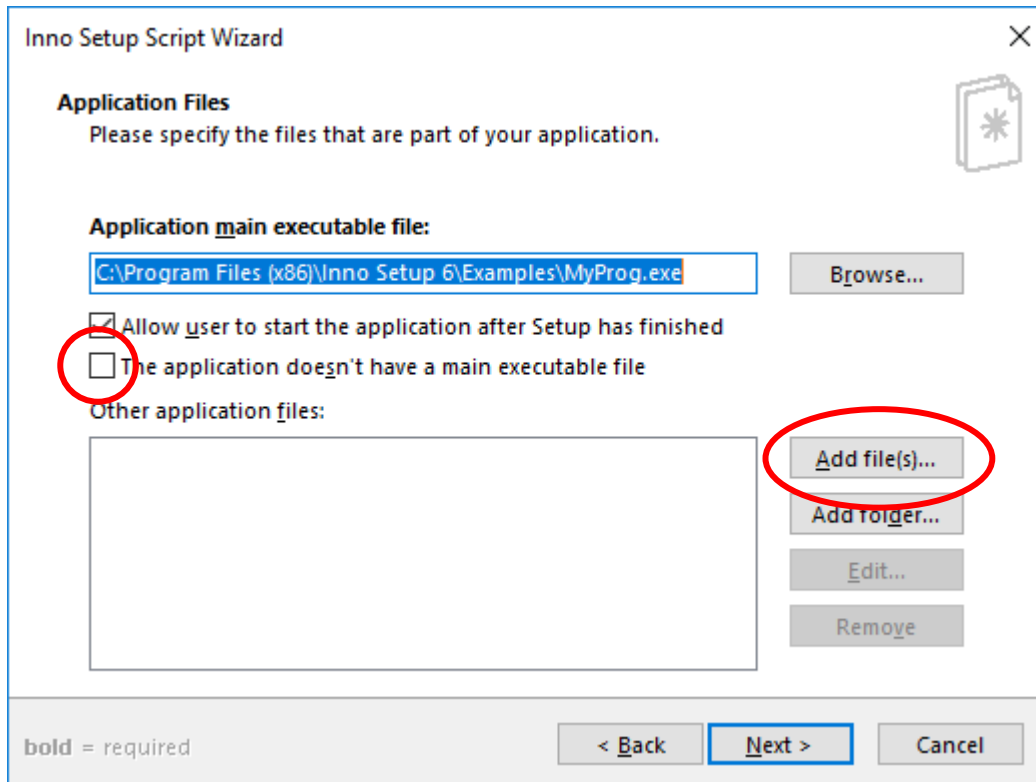
☐ The application doesn't need a folder

bold = required

< Back Next > Cancel

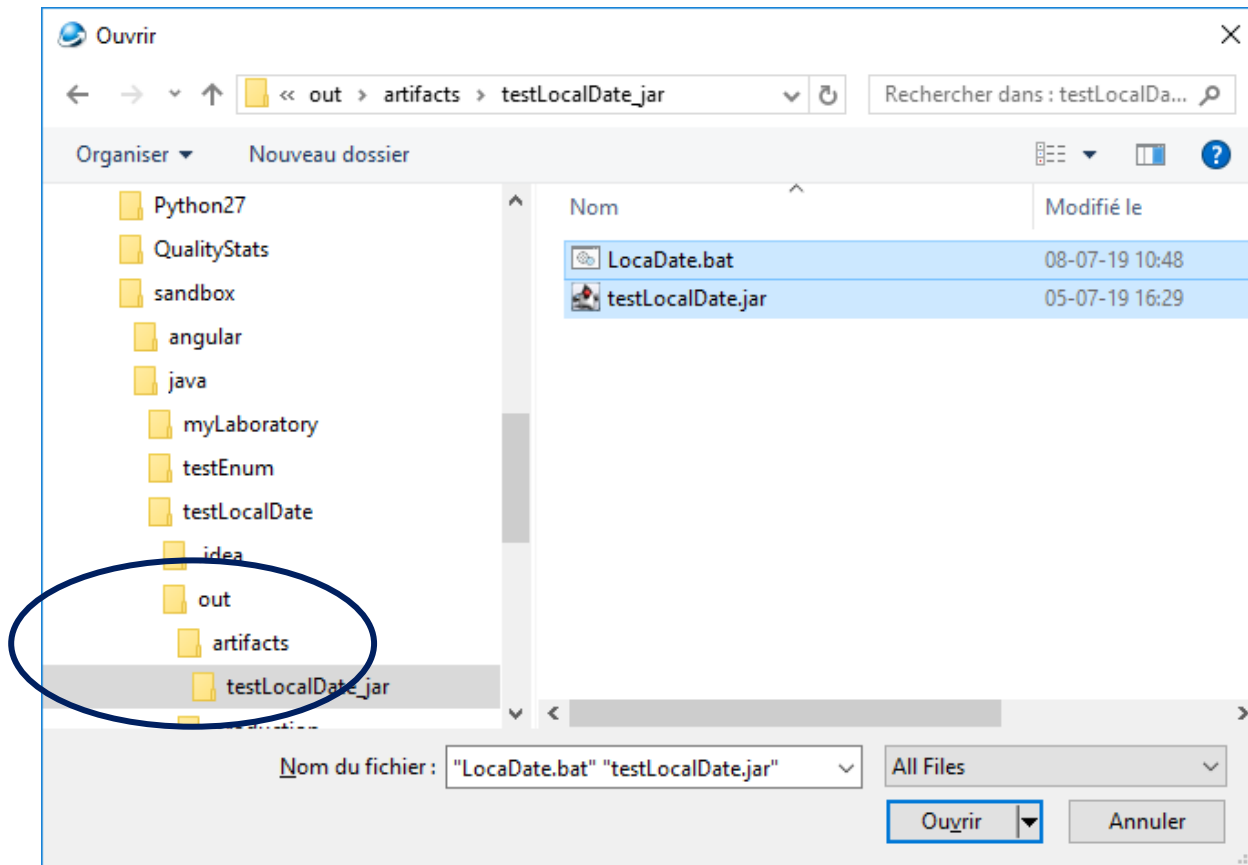
Comment créer un installateur de programme pour Windows ?

- InnoSetup



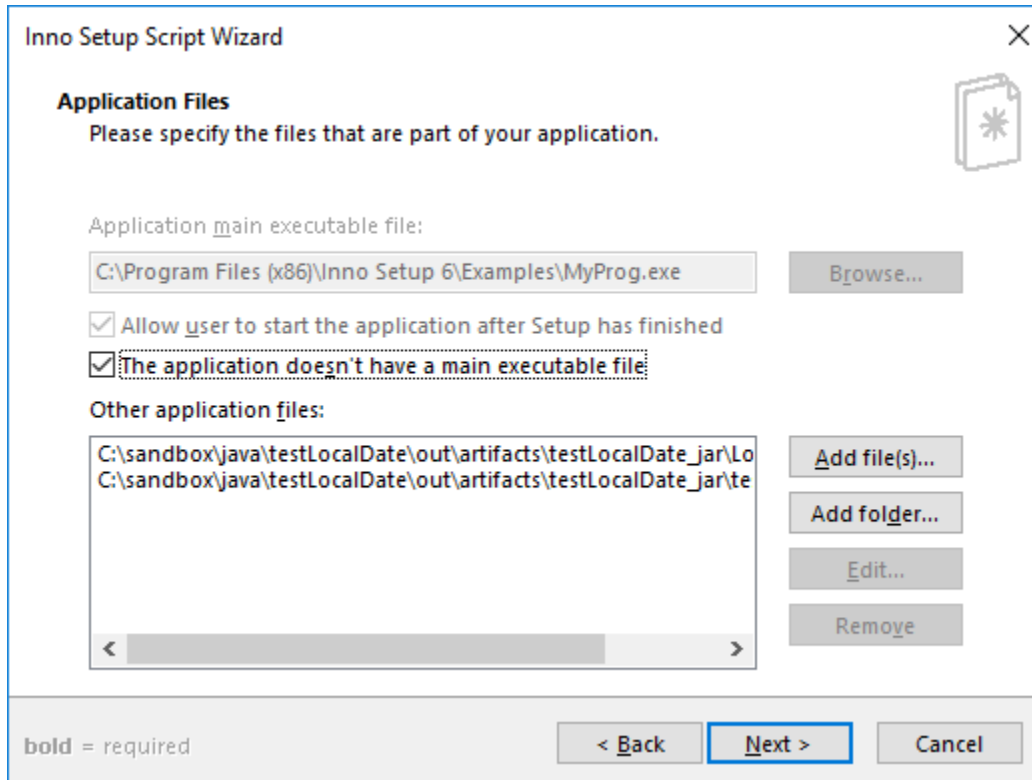
Comment créer un installateur de programme pour Windows ?

- InnoSetup



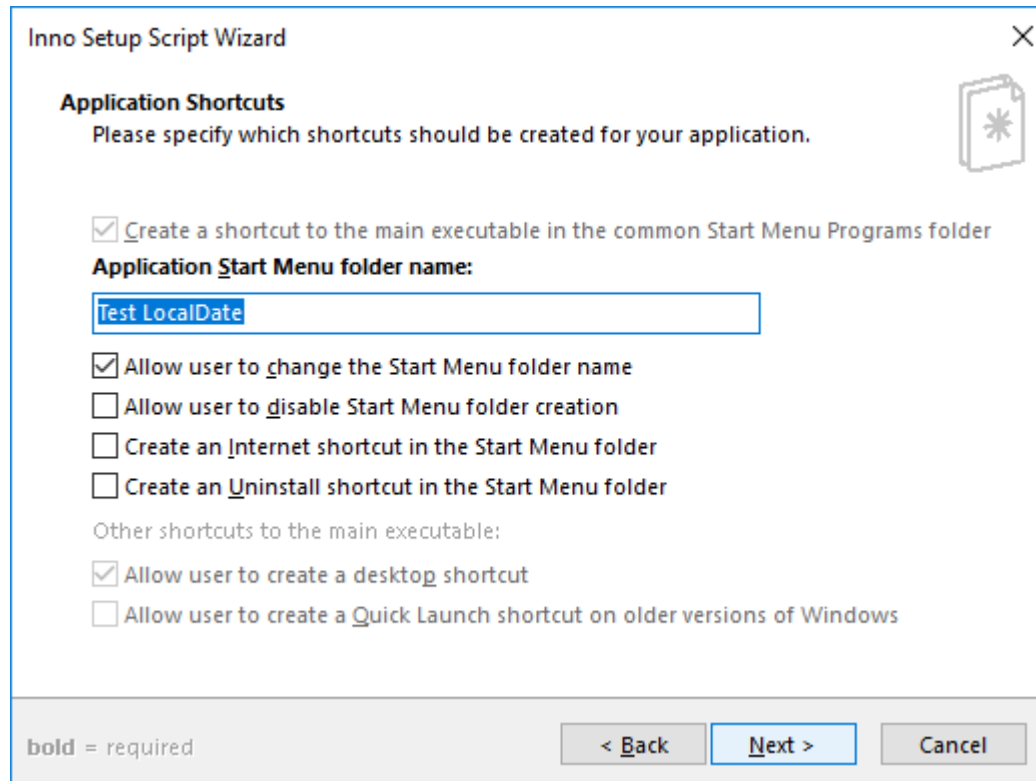
Comment créer un installateur de programme pour Windows ?

- InnoSetup



Comment créer un installateur de programme pour Windows ?

- InnoSetup



The screenshot shows the 'Inno Setup Script Wizard' window, specifically the 'Application Shortcuts' step. The window has a title bar with the text 'Inno Setup Script Wizard' and a close button. The main content area is titled 'Application Shortcuts' and contains the instruction 'Please specify which shortcuts should be created for your application.' Below this, there are several checkboxes and a text input field. The first checkbox is checked and labeled 'Create a shortcut to the main executable in the common Start Menu Programs folder'. Below it is a text input field labeled 'Application Start Menu folder name:' with the text 'Test LocalDate' entered. There are four more checkboxes: 'Allow user to change the Start Menu folder name' (checked), 'Allow user to disable Start Menu folder creation' (unchecked), 'Create an Internet shortcut in the Start Menu folder' (unchecked), and 'Create an Uninstall shortcut in the Start Menu folder' (unchecked). Below these is the text 'Other shortcuts to the main executable:' followed by two more checkboxes: 'Allow user to create a desktop shortcut' (checked) and 'Allow user to create a Quick Launch shortcut on older versions of Windows' (unchecked). At the bottom of the window, there is a legend 'bold = required' and three buttons: '< Back', 'Next >', and 'Cancel'.

Inno Setup Script Wizard

Application Shortcuts
Please specify which shortcuts should be created for your application.

☒ Create a shortcut to the main executable in the common Start Menu Programs folder

Application Start Menu folder name:
Test LocalDate

☒ Allow user to change the Start Menu folder name

☐ Allow user to disable Start Menu folder creation

☐ Create an Internet shortcut in the Start Menu folder

☐ Create an Uninstall shortcut in the Start Menu folder

Other shortcuts to the main executable:

☒ Allow user to create a desktop shortcut

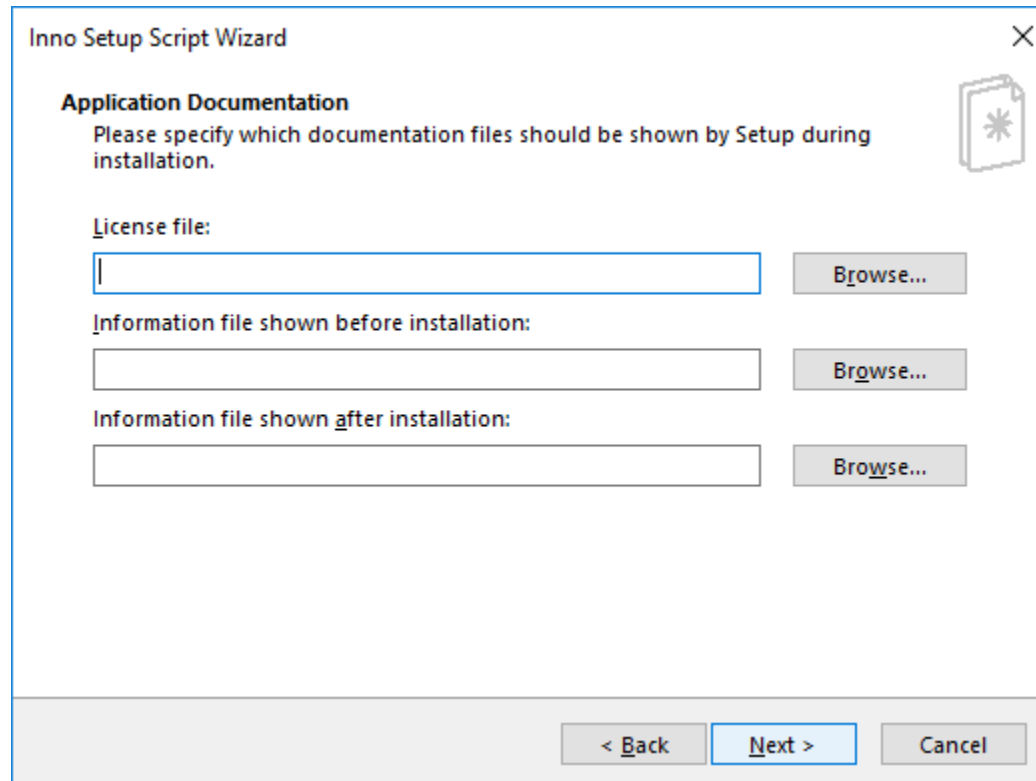
☐ Allow user to create a Quick Launch shortcut on older versions of Windows

bold = required

< Back Next > Cancel

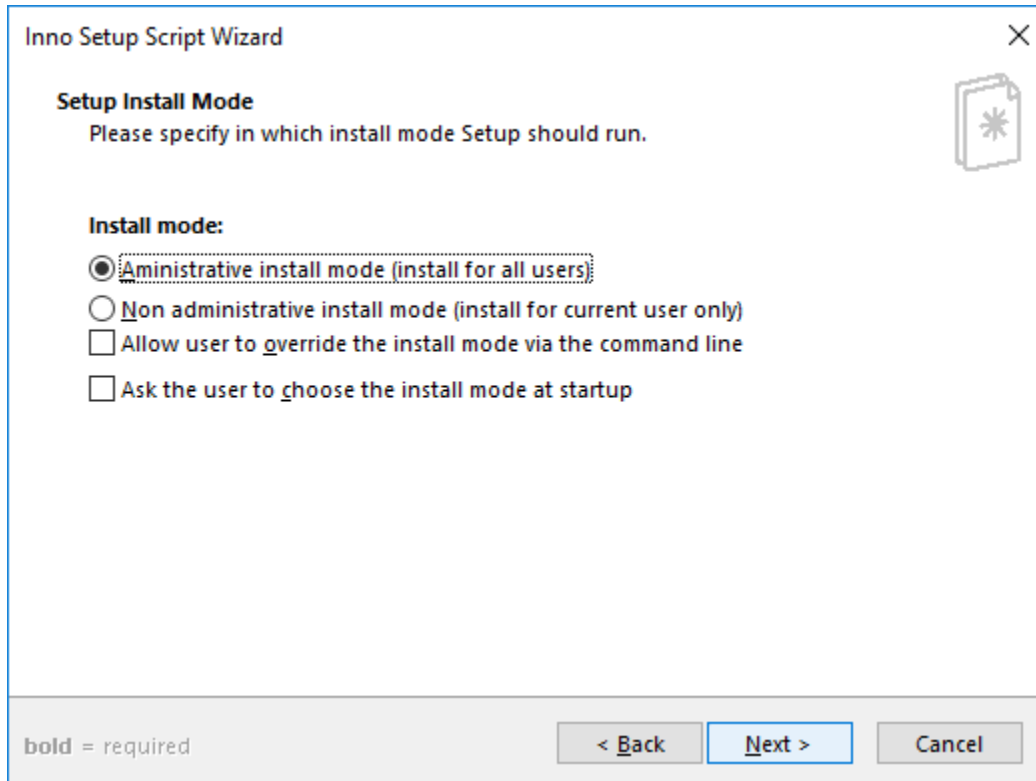
Comment créer un installateur de programme pour Windows ?

- InnoSetup



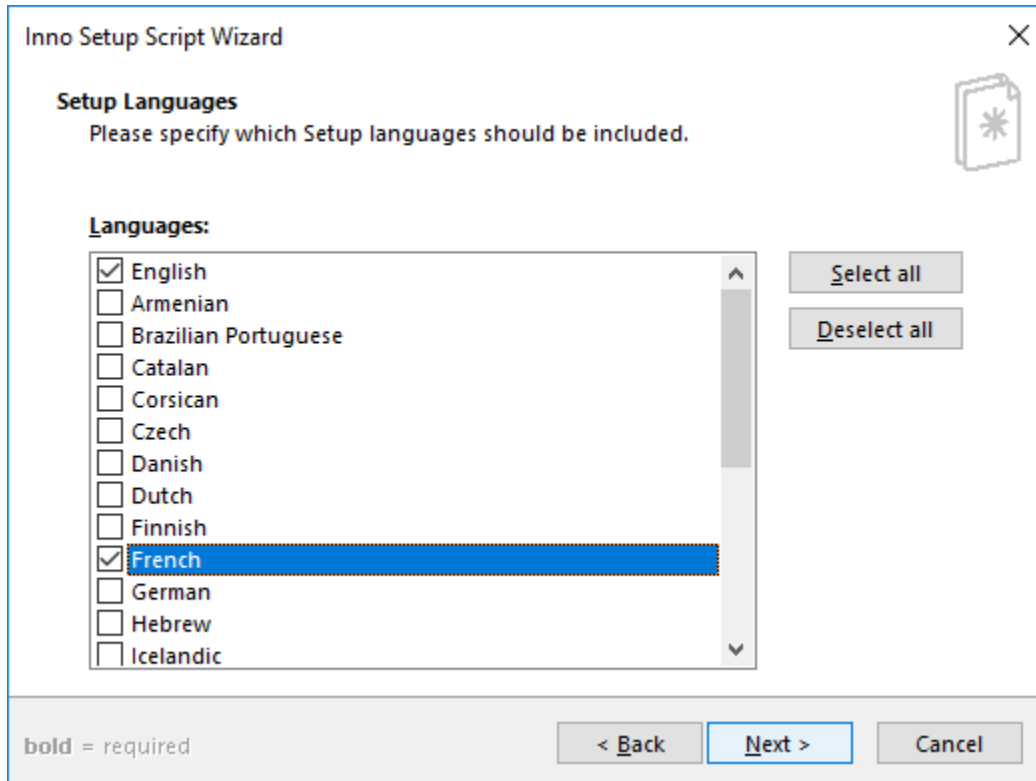
Comment créer un installateur de programme pour Windows ?

- InnoSetup



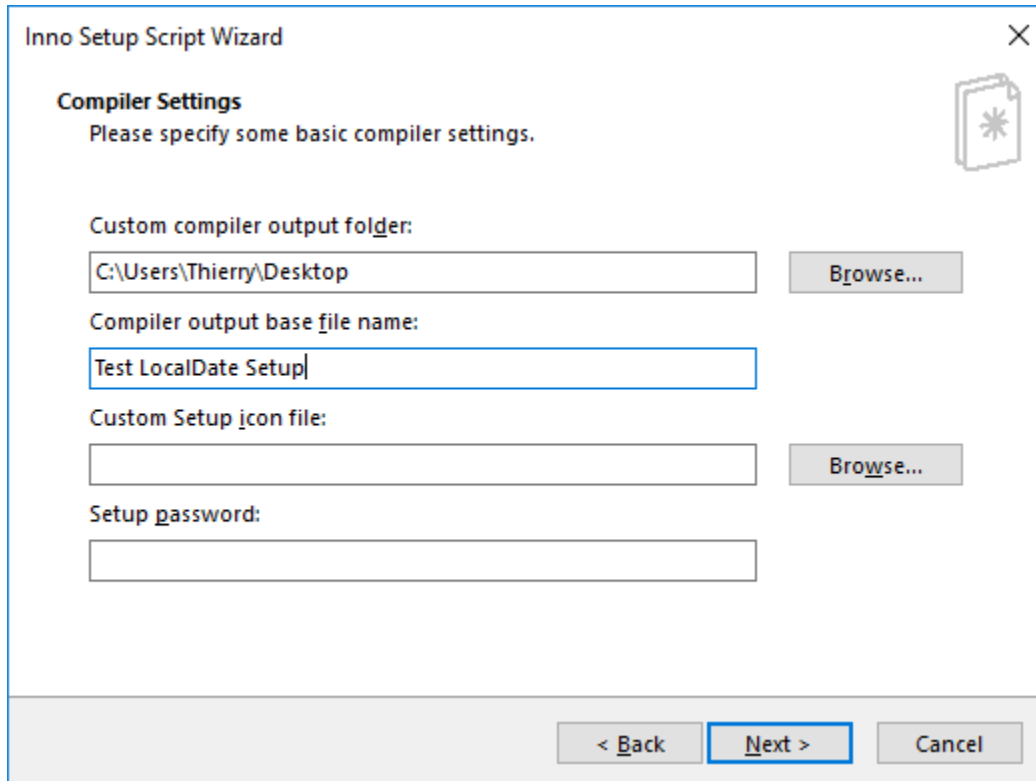
Comment créer un installateur de programme pour Windows ?

- InnoSetup



Comment créer un installateur de programme pour Windows ?

- InnoSetup



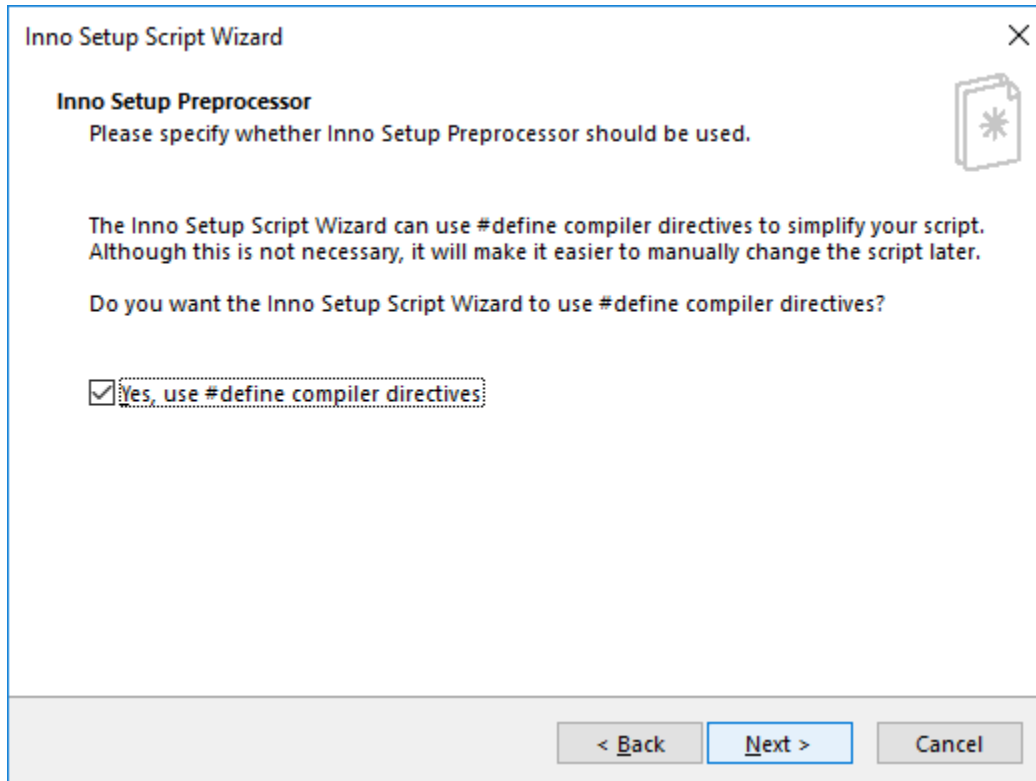
The image shows a screenshot of the 'Inno Setup Script Wizard' dialog box, specifically the 'Compiler Settings' step. The window has a title bar with the text 'Inno Setup Script Wizard' and a close button (X) in the top right corner. Below the title bar, the text 'Compiler Settings' is displayed, followed by the instruction 'Please specify some basic compiler settings.' To the right of this text is a small icon of a document with a star. The dialog contains four input fields with corresponding labels and buttons:

- Custom compiler output folder:** The input field contains 'C:\Users\Thierry\Desktop'. To its right is a 'Browse...' button.
- Compiler output base file name:** The input field contains 'Test LocalDate Setup'.
- Custom Setup icon file:** The input field is empty. To its right is a 'Browse...' button.
- Setup password:** The input field is empty.

At the bottom of the dialog, there are three buttons: '< Back', 'Next >', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

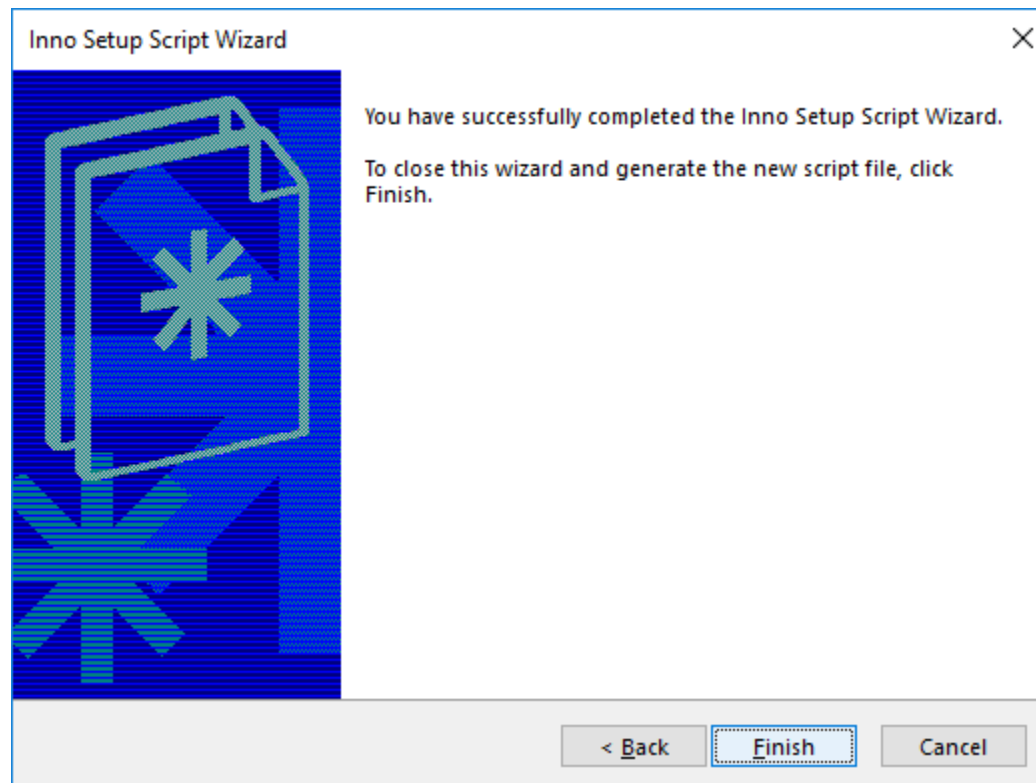
Comment créer un installateur de programme pour Windows ?

- InnoSetup



Comment créer un installateur de programme pour Windows ?

- InnoSetup



Comment créer un installateur de programme pour Windows ?

- InnoSetup

