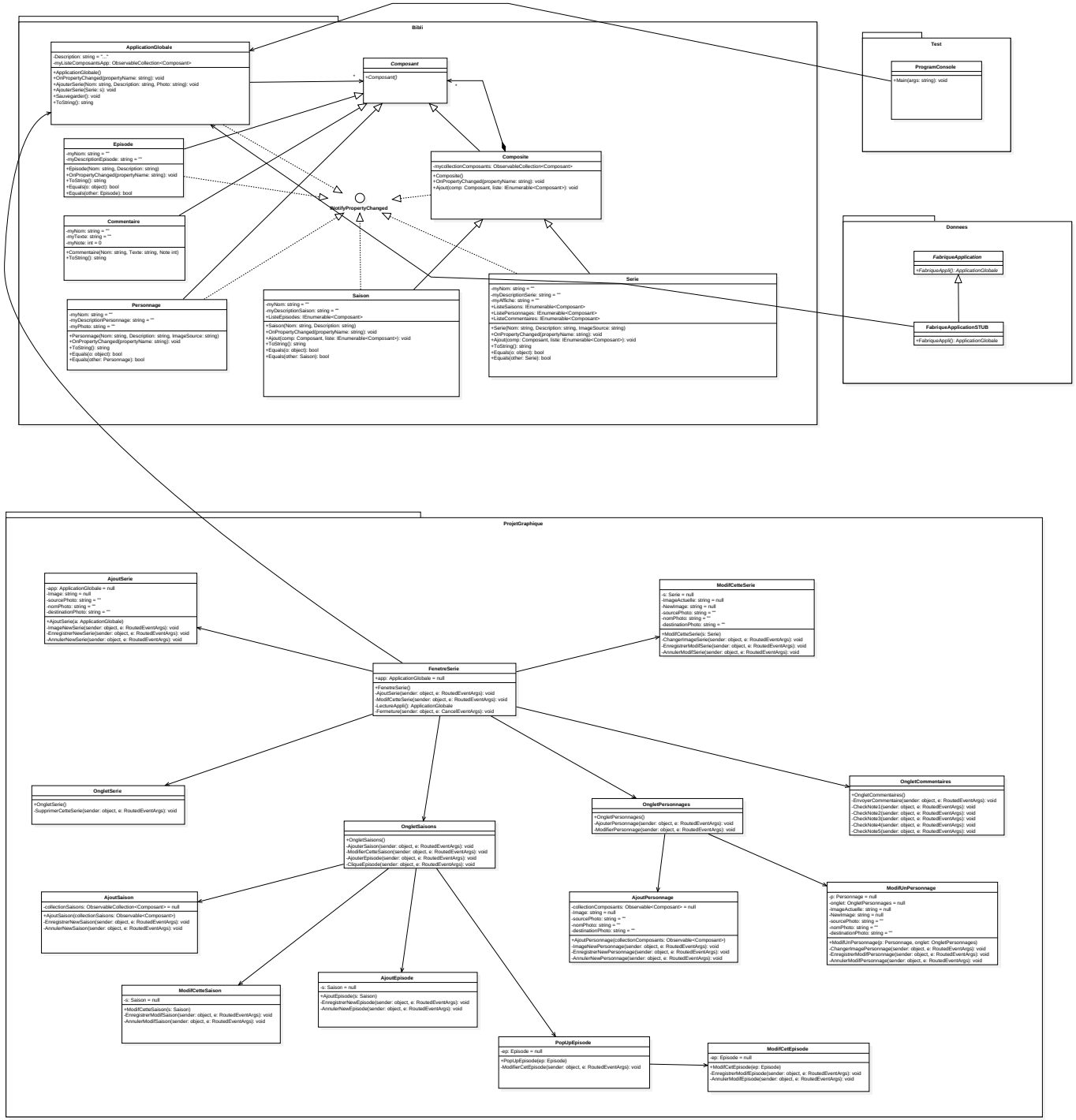
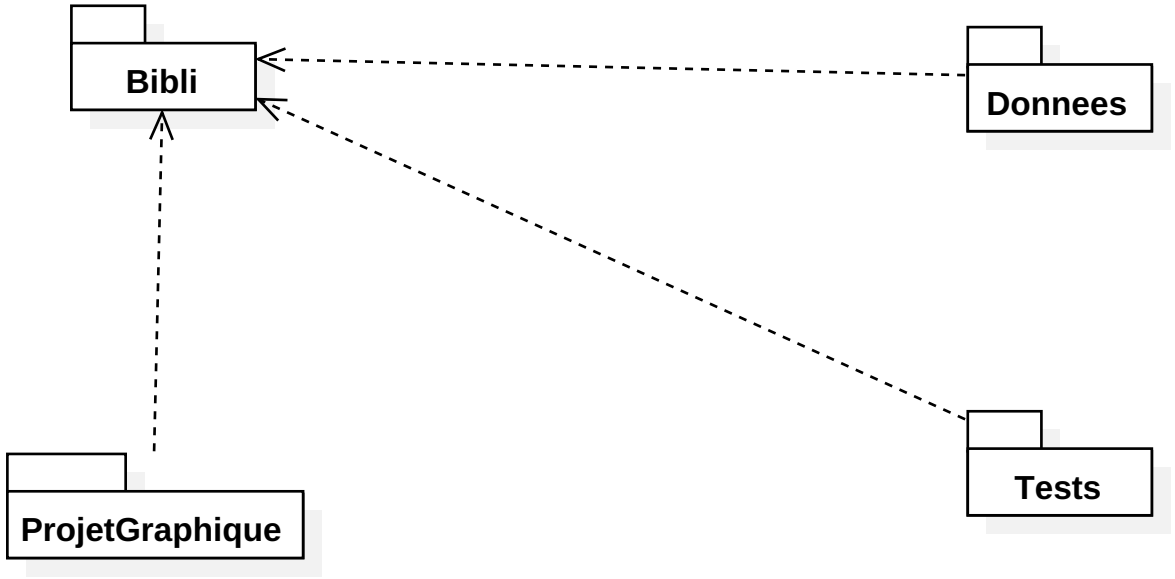
Model::Main


Bibli
ApplicationGlobale
-Description: string = "..."
-myListeComposantsApp: ObservableCollection<Composant>
+ApplicationGlobale()
+OnPropertyChanged(propertyName: string): void
+AjouterSerie(Nom: string, Description: string, Photo: string): void
+AjouterSerie(Serie: s): void
+Sauvegarder(): void
+ToString(): string
Composant
+Composant()
Test
ProgramConsole
+Main(args: string): void
Episode
-myNom: string = ""
-myDescriptionEpisode: string = ""
+Episode(Nom: string, Description: string)
+OnPropertyChanged(propertyName: string): void
+ToString(): string
+Equals(o: object): bool
+Equals(other: Episode): bool
Composite
-mycollectionComposants: ObservableCollection<Composant>
+Composite()
+OnPropertyChanged(propertyName: string): void
+Ajout(comp: Composant, liste: IEnumerable<Composant>): void
NotifyPropertyChanged
Commentaire
-myNom: string = ""
-myTexte: string = ""
-myNote: int = 0
+Commentaire(Nom: string, Texte: string, Note int)
+ToString(): string
Donnees
FabriqueApplication
+FabriqueAppli(): ApplicationGlobale
FabriqueApplicationSTUB
+FabriqueAppli(): ApplicationGlobale
Personnage
-myNom: string = ""
-myDescriptionPersonnage: string = ""
-myPhoto: string = ""
+Personnage(Nom: string, Description: string, ImageSource: string)
+OnPropertyChanged(propertyName: string): void
+ToString(): string
+Equals(o: object): bool
+Equals(other: Personnage): bool
Saison
-myNom: string = ""
-myDescriptionSaison: string = ""
+ListeEpisodes: IEnumerable<Composant>
+Saison(Nom: string, Description: string)
+OnPropertyChanged(propertyName: string): void
+Ajout(comp: Composant, liste: IEnumerable<Composant>): void
+ToString(): string
+Equals(o: object): bool
+Equals(other: Saison): bool
Serie
-myNom: string = ""
-myDescriptionSerie: string = ""
-myAffiche: string = ""
+ListeSaisons: IEnumerable<Composant>
+ListePersonnages: IEnumerable<Composant>
+ListeCommentaires: IEnumerable<Composant>
+Serie(Nom: string, Description: string, ImageSource: string)
+OnPropertyChanged(propertyName: string): void
+Ajout(comp: Composant, liste: IEnumerable<Composant>): void
+ToString(): string
+Equals(o: object): bool
+Equals(other: Serie): bool
ProjetGraphique
AjoutSerie
-app: ApplicationGlobale = null
-Image: string = null
-sourcePhoto: string = ""
-nomPhoto: string = ""
-destinationPhoto: string = ""
+AjoutSerie(a: ApplicationGlobale)
-ImageNewSerie(sender: object, e: RoutedEventArgs): void
-EnregistrerNewSerie(sender: object, e: RoutedEventArgs): void
-AnnulerNewSerie(sender: object, e: RoutedEventArgs): void
ModifCetteSerie
-s: Serie = null
-ImageActuelle: string = null
-NewImage: string = null
-sourcePhoto: string = ""
-nomPhoto: string = ""
-destinationPhoto: string = ""
+ModifCetteSerie(s: Serie)
-ChangerImageSerie(sender: object, e: RoutedEventArgs): void
-EnregistrerModifSerie(sender: object, e: RoutedEventArgs): void
-AnnulerModifSerie(sender: object, e: RoutedEventArgs): void
FenetreSerie
+app: ApplicationGlobale = null
+FenetreSerie()
-AjoutSerie(sender: object, e: RoutedEventArgs): void
-ModifCetteSerie(sender: object, e: RoutedEventArgs): void
-LectureAppli(): ApplicationGlobale
-Fermeture(sender: object, e: CancelEventArgs): void
OngletSerie
+OngletSerie()
-SupprimerCetteSerie(sender: object, e: RoutedEventArgs): void
OngletSaisons
+OngletSaisons()
-AjouterSaison(sender: object, e: RoutedEventArgs): void
-ModifierCetteSaison(sender: object, e: RoutedEventArgs): void
-AjouterEpisode(sender: object, e: RoutedEventArgs): void
-CliqueEpisode(sender: object, e: RoutedEventArgs): void
OngletPersonnages
+OngletPersonnages()
-AjouterPersonnage(sender: object, e: RoutedEventArgs): void
-ModifierPersonnage(sender: object, e: RoutedEventArgs): void
OngletCommentaires
+OngletCommentaires()
-EnvoyerCommentaire(sender: object, e: RoutedEventArgs): void
-CheckNote1(sender: object, e: RoutedEventArgs): void
-CheckNote2(sender: object, e: RoutedEventArgs): void
-CheckNote3(sender: object, e: RoutedEventArgs): void
-CheckNote4(sender: object, e: RoutedEventArgs): void
-CheckNote5(sender: object, e: RoutedEventArgs): void
AjoutSaison
-collectionSaisons: ObservableCollection<Composant> = null
+AjoutSaison(collectionSaisons: Observable<Composant>)
-EnregistrerNewSaison(sender: object, e: RoutedEventArgs): void
-AnnulerNewSaison(sender: object, e: RoutedEventArgs): void
AjoutPersonnage
-collectionComposants: Observable<Composant> = null
-Image: string = null
-sourcePhoto: string = ""
-nomPhoto: string = ""
-destinationPhoto: string = ""
+AjoutPersonnage(collectionComposants: Observable<Composant>)
-ImageNewPersonnage(sender: object, e: RoutedEventArgs): void
-EnregistrerNewPersonnage(sender: object, e: RoutedEventArgs): void
-AnnulerNewPersonnage(sender: object, e: RoutedEventArgs): void
ModifUnPersonnage
-p: Personnage = null
-onglet: OngletPersonnages = null
-ImageActuelle: string = null
-NewImage: string = null
-sourcePhoto: string = ""
-nomPhoto: string = ""
-destinationPhoto: string = ""
+ModifUnPersonnage(p: Personnage, onglet: OngletPersonnages)
-ChangerImagePersonnage(sender: object, e: RoutedEventArgs): void
-EnregistrerModifPersonnage(sender: object, e: RoutedEventArgs): void
-AnnulerModifPersonnage(sender: object, e: RoutedEventArgs): void
ModifCetteSaison
-s: Saison = null
+ModifCetteSaison(s: Saison)
-EnregistrerModifSaison(sender: object, e: RoutedEventArgs): void
-AnnulerModifSaison(sender: object, e: RoutedEventArgs): void
AjoutEpisode
-s: Saison = null
+AjoutEpisode(s: Saison)
-EnregistrerNewEpisode(sender: object, e: RoutedEventArgs): void
-AnnulerNewEpisode(sender: object, e: RoutedEventArgs): void
PopUpEpisode
-ep: Episode = null
+PopUpEpisode(ep: Episode)
-ModifierCetEpisode(sender: object, e: RoutedEventArgs): void
ModifCetEpisode
-ep: Episode = null
+ModifCetEpisode(ep: Episode)
-EnregistrerModifEpisode(sender: object, e: RoutedEventArgs): void
-AnnulerModifEpisode(sender: object, e: RoutedEventArgs): void

Model1::PackageDiagram1



Explications du diagramme de classes

Bibliothèque de classes « Bibli » :

ApplicationGlobale : Le but de la classe ApplicationGlobale est qu'elle est la Classe mère de l'application, elle contient comme attributs la description de l'application, et l'ObservableCollection de Composants qui contient tous les Composants de l'application. Elle sera générée lors du lancement de l'application.

- ApplicationGlobale () : Le constructeur de ApplicationGlobale qui ne prend rien comme arguments et qui renvoie une ApplicationGlobale.
- OnPropertyChanged (propertyName : string) : void : Méthode qui permet d'envoyer le fait que la liste des composants a changée avec un évènement.
- AjouterSerie (Nom : string, Description : string, Photo : string) : void : Méthode qui nous permet de rajouter une série à l'application. En utilisant les attributs de la série passée en paramètre.
- AjouterSerie (Série : s) : void : Méthode qui nous permet de rajouter une série à l'application. En utilisant une instance de Serie.
- Sauvegarde () : void : Méthode qui nous permet de sauvegarder l'application.
- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance d'application (Utilisée pour le ProgramConsole).

Episode : Cette classe représente la classe des épisodes de la série. Elle contient 2 attributs, elle a un Nom (string) et une Description (string).

- Episode (Nom : string, Description : string) : Constructeur de la classe Episode, elle prend un Nom et une Description comme paramètres.
- OnPropertyChanged (propertyName : string) : void : Méthode qui permet d'envoyer le fait que la liste des composants a changé avec un évènement.
- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance d'épisode (Utilisé pour le ProgramConsole).
- Equals (o : object) : bool : Permet de comparer deux instances d'épisode pour voir si elles sont de la même classe ou pas. Si elles sont pareilles elle appelle Equals (other : Episode) : bool. Ou si les deux instances sont exactement les mêmes elle renvoie un boolean avec 'true'. Et si les deux instances ne sont pas de la même classe elle renvoie un boolean 'false'.
- Equals (other : Episode) : bool : Regarde si les 2 épisodes ont les mêmes noms. Renvoie un Boolean avec la réponse.

Commentaire : Cette Classe représente la classe des commentaires de la série. Elle contient 3 Attributs, le Nom de la personne ayant mis le commentaire (string), le Texte du commentaire (string) et la Note que la personne a attribuée à la série (int).

- Commentaire (nom : string, texte : string, Note int) : Constructeur de la classe Commentaire elle prend un Nom, un Texte et une Note en paramètres. Et elle crée un Commentaire.

- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance de Commentaire. (Utilisé pour le ProgramConsole).

Personnage : Cette classe représente la classe des Personnages de la série. Elle contient 3 Attributs. Un Nom du personnage (string), une Description (string), et la Source de son image (string).

- Personnage (Nom : string, Description : string, ImageSource : string) : void : Constructeur de la classe Personnage, elle prend comme paramètres un Nom, une Description et une Source d'image. Elle renvoie une instance de Personnage.
- OnPropertyChanged (propertyName : string) : void : Méthode qui permet d'envoyer le fait que la liste des composants a changé avec un évènement.
- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance de Personnage (Utilisé pour le ProgramConsole).
- Equals (o : object) : bool : Permet de comparer deux instances de Personnages pour voir si elles sont de la même classe ou pas. Si elles sont pareilles elle appelle Equals (other : Personnage) : bool. Ou si les deux instances sont exactement les mêmes, elle renvoie un boolean avec true. Et si les deux instances ne sont pas de la même classe elle renvoie un boolean false.
- Equals (other : Personnage) : bool : Regarde si les 2 personnages ont les mêmes noms. Renvoie un boolean avec la réponse.

Saison : Cette classe représente la classe des Saisons de la série. Elle contient 3 attributs : un Nom, une Description et une liste (IEnumerable) d'épisodes (Composant).

- Saison (Nom : string, Description : string) : Constructeur de la classe Saison, elle prend 3 paramètres : un Nom, et une Description et elle crée une Saison.
- Ajout (comp : Composant, liste : IEnumerable<Composant>) : void : Méthode qui permet l'ajout d'un composant à la liste de Saison, dans ce cas là, l'ajout est un épisode.
- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance de Saison (Utilisé pour le ProgramConsole).
- Equals (o : object) : bool : Permet de comparer deux instances de Saison pour voir si elles sont de la même classe ou pas. Si elles sont pareilles elle appelle Equals (other : Saisons) : bool. Ou si les deux instances sont exactement les mêmes elle renvoie un boolean avec true. Et si les deux instances ne sont pas de la même classe elle renvoie un boolean false.
- Equals (other : Saison) : bool : Regarde si les 2 saison ont les mêmes Noms. Renvoie un boolean avec la réponse.

Serie : Cette Classe représente la classe des séries de l'application. Elle contient 6 attributs : un Nom, une Description, une Affiche (qui est un string avec la source de l'image), une ListeSaisons (IEnumerable) de Composants, une ListePersonnages (IEnumerable) de Composants et une ListeCommentaires (IEnumerable) de Composants.

- Serie (Nom : string, Description : string, ImageSource: string) : constructeur de la classe Serie, elle prend un Nom, une Description et une ImageSource (string) comme paramètres et elle crée une nouvelle Serie.
- Ajout (comp : Composant, liste : IEnumerable<Composant>) : void : Méthode qui permet l'ajout d'un composant à une des listes de Serie, dans ce cas-là l'ajout peut être une Saison, un Personnage ou un Commentaire. La méthode vérifie à quelle classe le composant appartient, et ensuite le rajoute dans la bonne liste.
- ToString () : void : Méthode qui nous permet de renvoyer les informations d'une instance de Serie (Utilisé pour le ProgramConsole).
- Equals (o : object) : bool : Permet de comparer deux instances de Serie pour voir si elles sont de la même classe ou pas. Si elles sont pareilles elle appelle Equals (other : Serie) : bool. Ou si les deux instances sont exactement les mêmes elle renvoie un boolean avec true. Et si les deux instances ne sont pas de la même classe elle renvoie un boolean false.
- Equals (other : Serie) : bool : Regarde si les 2 séries ont les mêmes Noms. Renvoie un boolean avec la réponse.

Composite : Classe qui hérite de Composant, elle a 1 attribut qui est l'ObservableCollection de Composants.

- Composite () : Constructeur de la classe Composite, elle ne prend rien comme paramètres et crée un composite.
- OnPropertyChanged (propertyName : string) : void : Méthode qui permet d'envoyer le fait que la liste des composants à changé avec un évènement.
- Ajout (comp : Composant, liste : IEnumerable<Composant>) : void : c'est dans cette méthode de la classe Composite que les composants sont rajoutés. Car toutes les classes qui implémentent « Ajout () » héritent de Composite.

Composant : Classe abstraite qui représente tous les composants de l'application. Toutes les différentes classes sauf ApplicationGlobale. Ceci est fait pour que l'application ne contienne qu'une seule grande liste et pas plusieurs petites listes.

- Composant () : Constructeur de Composant, elle ne prend pas de paramètres et elle crée un Composant.

Projet « Test » :

ProgramConsole : Le programme qui représente notre application et qui permet de vérifier que le métier fonctionne.

- Main(args string[]) : void : Fonction principale de ProgramConsole qui permet de faire fonctionner ProgramConsole.

Bibliothèque de classes « Donnees » : Ce projet permet d'avoir un « STUB »

FabriqueApplication : Classe abstraite

- FabriqueAppli () : Méthode abstraite de FabriqueApplication qui sera modifiée dans la classe fille FabriqueApplicationSTUB

FabriqueApllicationSTUB : Classe fille de FabriqueApplication et qui implémente sa méthode

- FabriqueAppli () : ApplicationGlobale : Méthode qui renvoie une application remplie de données (séries, saisons, épisodes, personnages, commentaires)

Projet « ProjetGraphique » :

FenêtreSerie : La fenêtre principale qui contient la « ListView » qui est le 1er Master de l'application. Elle a comme attribut une ApplicationGlobale.

- FenetreSerie () : crée la FenetreSerie.
- AjoutSerie (sender : object, e : RoutedEventArgs) : void : fonction qui est appelée quand on appuie sur le bouton « Ajouter une série ». Elle appelle la fenêtre AjoutSerie avec l'application globale comme paramètre.
- ModifCetteSerie (sender : object, e : RoutedEventArgs): void : fonction qui est appelée quand on appuie sur le bouton « Modifier cette série ». Elle appelle la fenêtre ModifCetteSerie avec l'application globale comme paramètre.
- LectureAppli () : ApplicationGlobale : Méthode qui lis le fichier de sauvegarde pour ensuite créer une application avec les informations du fichier.
- Fermeture (sender: object, e : RoutedEventArgs) : void : Evènement qui est appelé lors de la fermeture de l'application elle permet de demander à l'utilisateur s'il veut sauvegarder ses changements et appel Sauvegarder () si l'utilisateur valide.

ModifCetteSerie : Une fenêtre pop-up qui peut modifier une série.

- ModifCetteSerie (s : Serie) : crée la fenêtre ModifCetteSerie, elle reçoit la série qui doit être changée. Cette série deviendra la DataContext de ModifCetteSerie.
- ChangerImageSerie (sender : object, e:RoutedEventArgs) : void : Méthode qui permet à l'utilisateur de changer l'affiche de la série.
- EnregistrerModifSerie (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer les changements faits par l'utilisateur.
- AnnulerModifSerie (sender: object, e:RoutedEventArgs) : void : Annule les modifications faits et ferme le pop-up ModifCetteSerie.

AjoutSerie : Une fenêtre pop-up qui permet à l'utilisateur de rajouter une nouvelle série à la liste de séries.

- AjoutSerie (a : ApplicationGlobale) : crée la fenêtre AjoutSerie, elle reçoit l'ApplicationGlobale où elle rajoutera la nouvelle série.

- ImageNewSerie (sender: object, e:RoutedEventArgs) : void : méthode qui permet à l'utilisateur de rajouter une image à la série.
- EnregistrerNewSerie (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer la nouvelle série dans la liste de séries.
- AnnulerModifSerie (sender: object, e:RoutedEventArgs) : void : Annule les modifications faits et ferme le pop-up AjoutSerie.

OngletSerie : Un des 4 onglets (UserControl) qui sont dans le ContentControl de FenetreSerie. C'est dans cet onglet que seront affiché le nom et l'affiche de la série, ainsi qu'un bouton « Supprimer cette série ».

- OngletSerie () : crée l'onglet OngletSerie, il sera changé en fonction de quelle série est choisie dans le Master de FenetreSerie.
- SupprimerCetteSerie (sender: object, e:RoutedEventArgs) : fonction qui est appelée quand l'utilisateur appuie sur le bouton « Supprimer cette série ». Elle demande une vérification et si l'utilisateur valide, elle supprimera la série qui est choisie dans le Master de FenetreSerie.

OngletSaison : Un des 4 onglets (UserControl) qui sont dans le ContentControl de FenetreSerie. C'est dans cet onglet que sera affiché un Master avec la liste de toutes les saisons de la série. Le Detail de ce Master est composé de 2 parties. La partie supérieure représente une courte description de la série et la partie inferieure contient une ListView qui contient la liste de tous les épisodes de la saison choisie. Quand on clique sur un épisode elle ouvre PopUpEpisode

- AjouterSaison (sender: object, e:RoutedEventArgs) : void : Fonction qui est appelée lorsque l'utilisateur appuie sur le bouton « Ajouter une Saison » elle ouvre la fenêtre AjoutSaison
- ModifierCettesaison (sender: object, e:RoutedEventArgs) : void : Fonction qui est appelée lorsque l'utilisateur appuie sur le bouton « Modifier cette série ». Elle ouvre la fenêtre ModifCetteSaison et elle envoie aussi la saison sélectionnée dans la ListView.
- AjouterEpisode (sender: object, e:RoutedEventArgs) : void : Fonction qui est appelée lorsque l'utilisateur appuie sur le bouton « Ajouter un épisode ». Elle ouvre la fenêtre AjoutEpisode.

AjoutSaison : Une fenêtre pop-up qui permet à l'utilisateur de rajouter une nouvelle saison à la liste de Saison de la série.

- AjoutSaison (ObservableCollection<Composant>) : crée la fenêtre AjoutSerie, elle reçoit une ObservableCollection où elle rajoutera la nouvelle saison.
- EnregistrerNewSaison (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer la nouvelle saison dans la liste de Saisons.
- AnnulerNewSaison (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up AjoutSaison.

ModifCetteSaison : Une fenêtre pop-up qui peut modifier une Saison.

- ModifCetteSaison (s : Saison) : crée la fenêtre ModifCetteSaison, elle reçoit la saison qui doit être changée. Cette saison deviendra la DataContext de ModifCetteSaison.

EnregistrerModifSerie (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer les changements faits par l'utilisateur.

- AnnulerModifSerie (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up ModifCetteSaison.

AjouterEpisode (S : Saison) : crée la fenêtre AjouterEpisode, elle reçoit la saison dans laquelle elle ajoutera le nouvel épisode.

- EnregistrerNewSaison (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer le nouvel Episode dans la liste d'épisodes.
- AnnulerNewEpisode (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up AjouterEpisode.

PopUpEpisode : un pop-up qui est affiché lorsqu'on clique sur un épisode dans la liste des épisodes. Elle affiche la description de l'épisode ainsi que son nom.

- PopUpEpisode (ep : Episode) : crée la fenêtre PopUpEpisode, elle reçoit l'épisode qui a été cliqué dans la ListView. Cet épisode deviendra le DataContext du PopUpEpisode.
- ModifierCetEpisode (sender: object, e:RoutedEventArgs) : void ; Méthode qui est appelée lorsqu'on clique sur le bouton « Modifier cet épisode ». Elle appelle la Fenêtre ModifCetEpisode et elle envoie l'épisode du PopUpEpisode

ModifCetEpisode : un autre pop-up qui est affiche et qui permet à l'utilisateur de changer les informations de l'épisode.

- ModifCetteSaison (s : Saison) : crée la fenêtre ModifCetEpisode, elle reçoit l'épisode qui doit être changé. Cet épisode deviendra la DataContext de ModifCetEpisode.
- EnregistrerModifSerie (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer les changements fait par l'utilisateur.
- AnnulerModifSerie (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up ModifCetEpisode.

OngletPersonnages : Un des 4 onglets (UserControl) qui sont dans le ContentControl de FenetreSerie. C'est dans cet onglet que sera affichée la liste de tous les personnages de la série dans un tableau. Ce tableau contient leurs noms, leurs photos et une courte description. Le tableau est fait à partir d'un DataGrid.

- AjouterPersonnage (sender: object, e:RoutedEventArgs) : void : Fonction qui est appelée lorsque l'utilisateur appuie sur le bouton « Ajouter un Personnage ». Elle ouvre la fenêtre AjouterPersonnage
- ModifierPersonnage (sender: object, e:RoutedEventArgs) : void : Fonction qui est appelée lorsque l'utilisateur appuie sur le bouton « Modifier ce Personnage ». Elle ouvre la fenêtre ModifunPersonnage et elle envoie aussi le Personnage sélectionné dans le DataGrid.

AjoutPersonnage :

- AjoutPersonnage (ObservableCollection<Composant>) : crée la fenêtre AjoutPersonnage, elle reçoit l'ApplicationGlobale où elle rajoutera la nouvelle série.
- ImageNewSerie (sender: object, e:RoutedEventArgs) : void : méthode qui permet l'utilisateur de rajouter l'image du personnage.
- EnregistrerNewPersonnage (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer le nouveau personnage dans la liste de Personnages.
- AnnulerNewPersonnage (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up AjoutPersonnage.

ModifUnPersonnage : Une fenêtre pop-up qui peut modifier une série.

- ModifUnPersonnage (s : Serie) : crée la fenêtre ModifUnPersonnage, elle reçoit le personnage qui doit être changé. Ce personnage deviendra la DataContext de ModifUnPersonnage.
- ChangerImagePersonnage (sender : object, e:RoutedEventArgs) : void : Méthode qui permet à l'utilisateur de changer la photo du personnage.
- EnregistrerModifPersonnage (sender: object, e:RoutedEventArgs) : void : Méthode qui sert à enregistrer les changements faits par l'utilisateur.
- AnnulerModifPersonnage (sender: object, e:RoutedEventArgs) : void : Annule les modifications faites et ferme le pop-up ModifUnPersonnage.

OngletCommentaires : Un des 4 onglets (UserControl) qui sont dans le ContentControl de FenetreSerie. Cet onglet permet à l'utilisateur de rajouter un commentaire à la série, ainsi il peut y voir les commentaires que les autres utilisateurs ont laissés sur cette série.

- EnvoyerCommentaire (sender : object, e:RoutedEventArgs) : void : Méthode qui permet d'enregistrer le commentaire saisi par l'utilisateur. En appuyant sur le bouton « Envoyer », le commentaire sera enregistré dans la liste de commentaires et elle s'affichera dans le DataGrid d'en dessous qui contient tous les commentaires sur la série.
- CheckNote1 (sender:object, e:RoutedEventArgs): void : permet de donner une note de 1/5 à la Serie.
- CheckNote2 (sender:object, e:RoutedEventArgs): void : permet de donner une note de 2/5 à la Serie.
- CheckNote3 (sender:object, e:RoutedEventArgs): void : permet de donner une note de 3/5 à la Serie.
- CheckNote4 (sender:object, e:RoutedEventArgs): void : permet de donner une note de 4/5 à la Serie.
- CheckNote5 (sender:object, e:RoutedEventArgs): void : permet de donner une note de 5/5 à la Serie.

Explications du diagramme de paquetages

Paquetage « Bibli » :

Ce paquetage regroupe les différentes classes du notre projet. C'est le cœur du projet.

Paquetage « ProjetGraphique » :

Ce paquetage possède les différentes vues de notre application. Il doit être relié à la « Bibli » car on doit pouvoir afficher quelque chose. C'est pour cela que l'on crée des instances des différentes classes, et que l'on 'binde' le tout sur les vues.

Paquetage « Donnees » :

Ce paquetage sert seulement à regrouper le STUB du projet, que l'on utilisé lors des tests sur console et aussi pour remplir nos vus. Cela nous évite de réécrire plusieurs en dur le code permettant de remplir notre application.

Paquetage « Tests » :

Ce paquetage ne possède qu'une classe pour faire les tests sur console.