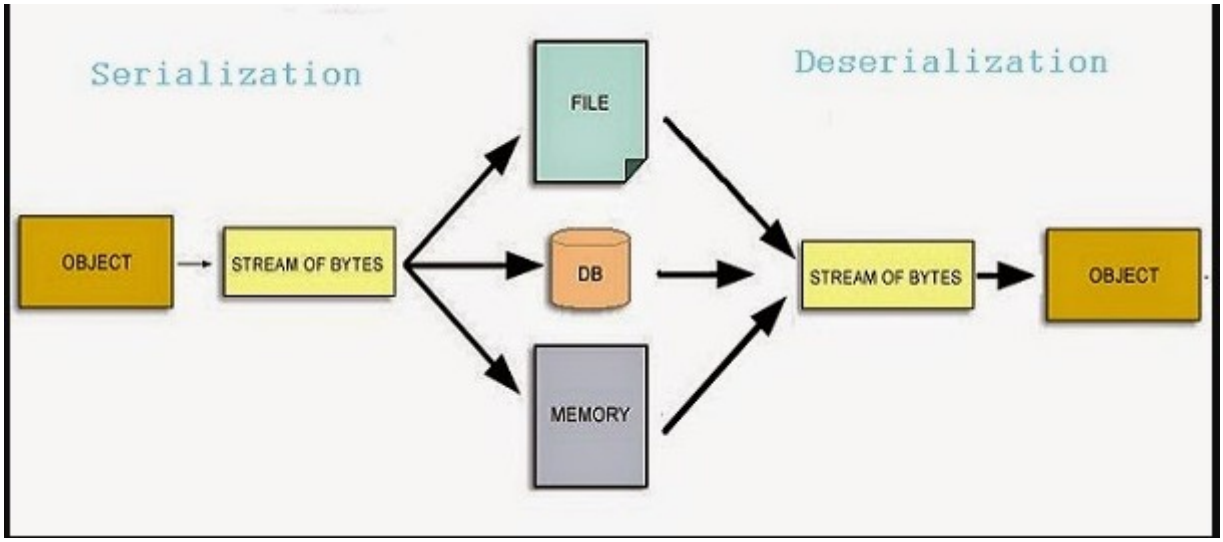
Documentation Serialization :

Dans notre projet, nous avons eu l'occasion d'utiliser la notion Java de Serialization. C'est à dire le stockage d'un objet java sous un format spécifique. Ainsi on transforme un objet en une suite de d'octets que l'on peut écrire dans un fichier et ainsi stocker l'objet java en dehors de l'execution. Enfin, en utilisant un socket d'ObjectInputStream, on peut de nouveau transformer une série d'octets en un objet utilisable.



Il faut faire attention, car un objet n'est pas Serializable de base. Il faut que la classe de l'objet implémente Serializable, mais également que cette classe dispose d'un serialversionUID qui se doit d'être un Long. Cet UID servira de clé lors de la création et la reconstruction d'un objet sérializer. Il permettra de s'assurer que les objets soit d'une même classe.

Un petit exemple : un objet UID 1 qui est déjà serializer et qui passe par la fonction readObject() d'une classe avec UID 2 remontera une erreur.

Pour écrire un objet serializable, il faut disposer d'un socket de sortie d'objet qui passera par la méthode writeObject() pour écrire un objet serializable.

A l'inverse, déserializer un objet demande un socket d'entrée d'objet qui passera par la méthode readObject() et transformera la série d'octets en un objet java utilisable.

Le gros avantage de la serialization étant que Java utilise un mecanisme qui permet de d'éviter de serializer plusieurs fois un même objet en relation de plusieurs autres objets. Chaque objet est associé à un numéro de série, qui est placé dans la relation, et n'est sérializé qu'une seule fois. De même, lors de la désérialization, chaque objet est désérializé une unique fois.