Les annotations peuvent se définir sur des classes, méthodes ou des attributs :

@Target(ElementType.TYPE) sur une classe

@Target(ElementType.METHOD) sur une méthode.

@Target(ElementType.FIELD) sur un attribut.

Généralement l’annotation se déclare comme une interface mais préfixé par @ :

Public @interface menu{ }

Une annotation pour qu’elle puisse être utiliser au cour de l’exécution par la VM , elle doit être annoter par :

@Retention(RetentionPolicy.RUNTIME) 🡪 signifie qu’on peut l’utiliser dans la reflexion.

L’annotation peut avoir un attribut mais il est déclaré comme une fonction :

String title()  default ‘’’’ 🡪 ici il s’agit d’un attribut title, si elle n’est pas spécifier il va être considérer comme une chaine vide.

Lorsqu’on a une annotation avec une seul attribut value(), on peut écrire directement sa valeur entre les parenthèse sans spécifier le nom de l’attribut 🡪 @Menu(‘’duTexteDeAttributValue’’)

Donc pour tirer profit d’une annotation il faut utiliser un engine qui est une classe qui utilise la reflexion et au moment de l’exécution on lit la classe et on vérifie s’il contient des annotations, si oui on utiliser les valeurs de ses annotations pour faire le traitement qu’on veut avec.

Reflexion :

Form fa = cls.getDeclaredAnnotation(Form.**class**); 🡪 permet d’avoir l’annotation de type Form dans la classe cls, s’elle n’existe pas ce fa sera null.

Field f[] = cls.getDeclaredFields(); 🡪 permet d’obtenir les attributs d’une classe au moment de l’execution.

Pour pouvoir utiliser la réflexion en java, il faut tout d'abord avoir l'objet class d'une classe :

Class reflectClass = UFOEnemyShip.**class** 🡪 permet d'avoir l'objet Class, pour l'utilisé dans la réflexion

String className = reflectClass.getName() 🡪 permet d'avoir le nom complètement qualifié de la classe

**int** classModifier = reflectClass.getModifiers() 🡪 permet d'avoir le modificateur de la classe en question encodé dans un Integer.

**boolean** isPublic = Modifier.*isPublic*(classModifier); 🡪 renvoie true dans le cas où la classe á un modificateur public.

Sur un modificateur on peut tester les choses suivantes : isAbstract, isFinal, isInterface, isPrivate, isProtected, isStatic, isStrict, isSynchronized, isVolatile.

Class[] interfaces = reflectClass.getInterfaces(); 🡪 permet d'avoir les interfaces déclarer dans la classe en question.

Class classSuper = reflectClass.getSuperclass(); 🡪 permet d'avoir l'objet classe de la classe mère de la classe en question.

Method[] methods = reflectClass.getMethods(); 🡪 permet d'avoir tous les méthodes de la classe en question dans un tableau.

method.getName() 🡪 avoir le nom du la méthode,

method.getReturnType(); 🡪 retourn un objet class de type de retour de la méthode en question.

Class[] parameters = method.getParameterTypes(); 🡪 permet d'avoir les classes des parametres de la méthode en question.

Constructor<?> constructor = reflectClass.getConstructor(**new** Class[] { Integer.**class**, String.**class** }); 🡪 ce code nous permettre de récupérer le constructeur, qui a comme paramètres un Integer et un String.

Object obj = constructor.newInstance(12, "Toto is a good Man"); 🡪 newInstance sur un objet consturctor nous permet d'avoir une instance de la classe de constructeur.

Field idCode = reflectClass.getDeclaredField("idcode"); 🡪 permet d'avoir un objet Field d'un champ de la classe en question.

idCode.setAccessible(**true**); 🡪 Appliquer sur un objet de Type Filed pour rendre ce champ public dans le cas où il est déclarer private.

String code = (String)idCode.get(obj); 🡪 méthode get reçoit un objet sur lequel on cherche la valeur de l'objet field (la valeur du champ idCode) en question.

Method m = reflectClass.getDeclaredMethod("getPrivate", **null**); 🡪 permet de récupérer un Objet Method qui représente la méthode qui a comme nom "getPrivate" et aucun paramètre (null) á partir de la classe en question.

m.invoke(obj, **null**); 🡪 permet d'invoquer la méthode m avec aucun paramètre (null) sur l'objet obj.