**TP3 Labyrinthe en mode graphique : semaines 19/3 et 2/4**

Table des matières

[1) Présentation du sujet 2](#_Toc508610017)

[2) Code en mode console à télécharger de campus 2](#_Toc508610018)

[3) Le pattern Modèle Vue Contrôleur du code 2](#_Toc508610019)

[3.1) Le package *modele* 3](#_Toc508610020)

[3.2) Le package *vue* 4](#_Toc508610021)

[3.2) Le package *controleur* 4](#_Toc508610022)

[TRAVAIL A FAIRE : 5](#_Toc508610023)

[Exercice 1 : conception du diagramme de classes 5](#_Toc508610024)

[Exercice 2 : implémentation du labyrinthe en mode graphique 5](#_Toc508610025)

[ANNEXE : extraits de la documentation API 7](#_Toc508610026)

[java.util 7](#_Toc508610027)

[Class Scanner 7](#_Toc508610028)

[java.util 8](#_Toc508610029)

[**Class ArrayList<E>** 8](#_Toc508610030)

[java.util 11](#_Toc508610031)

[**Class Random** 11](#_Toc508610032)

# Présentation du sujet

Un labyrinthe est enregistré dans un fichier texte, en respect du format ci-dessous :

**5 5 0 0 4 4** (taille du labyrinthe en X et Y, point de départ en X et Y, point d’arrivée en X et Y)

**\_X\_\_\_** (dessin du labyrinthe avec **X** un mur et **\_** une case à trou)

**\_XX\_\_**

**\_ XX \_\_**

**\_ XX \_\_**

**\_\_X\_\_**

Le fichier [labyrinthe.txt](http://campus.ece.fr/mod/resource/view.php?id=12956) sur la page campus « POO Java » est à télécharger. Vous pouvez le modifier en respect de son format ci-dessus.

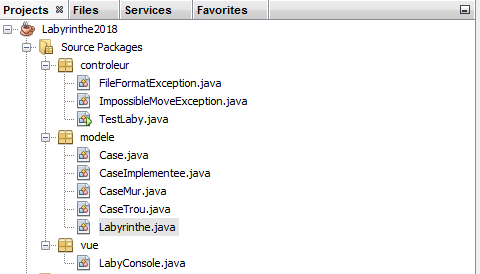
L’objectif est de charger le fichier dans des objets (en mémoire) du labyrinthe, de l’afficher et de se déplacer du point de départ au point d’arrivée en passant par les cases à trou **\_** et en évitant les murs **X**.

# Code en mode console à télécharger de campus

Le fichier Labyrinthe2018.zip contient le code de ce jeu en mode console, à télécharger et à dézipper de la page campus de « POO Java » <http://campus.ece.fr/course/view.php?id=124>. La **javadoc** (documentation en ligne) de ce code se trouve dans le dossier **dist** de ce fichier zip. Pour récupérer le code de ce fichier sur NetBeans, consulter le tutoriel [Mes premiers pas sur NetBeans](http://campus.ece.fr/mod/resource/view.php?id=5334).

# Le pattern Modèle Vue Contrôleur du code

Comme le montre la copie d’écran ci-dessous, ce code est structuré en 3 packages selon le pattern MVC *modele*, *vue* et *controleur*.



Pour plus d’informations sur le pattern MVC, consulter les sources suivantes :

<https://openclassrooms.com/courses/apprenez-a-programmer-en-java/mieux-structurer-son-code-le-pattern-mvc>

<https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur>

# 3.1) Le package *modele*

Le package *modele* contient les classes suivantes où se trouvent les données :

* Une interface *Case* représente une case du labyrinthe avec les prototypes des méthodes (getters et setters) suivantes :

*public interface Case {*

*public int getPositionX() ; // retourne la position en X (colonne) de la case*

*public int getPositionY() ; // retourne la position en Y (ligne) de la case*

*public boolean getVisited(); // retourne un booléen indiquant si la case est visitée ou non*

*public void setVisited(); // modifie la case pour qu’elle soit visitée*

*}*

* Une classe *CaseImplementee* implémente l’interface Case et toutes ses méthodes. Ses attributs protégés sont
* *posX* et *posY* : position en X (colonne) et Y (ligne) de la case,
* *visited* : booléen qui indique si la case est visitée ou non
* Les classes *CaseMur* et *CaseTrou* héritent de la classe *CaseImplementee*.
* Une classe *Labyrinthe* contient en attribut une grille composée de cases : *ArrayList<ArrayList<Case>> grille*. Ses dimensions en X (largeur) et en Y (hauteur) sont définies par les attributs *tailleX* et *tailleY*. Le point de départ est donné par les attributs *departX* et *departY*, l’arrivée par les attributs *arriveeX* et *arriveeY*, la position courante par les attributs *posX* et *posY*. Tous les attributs sont privés. La classe *Labyrinthe* contient entre autres les méthodes suivantes :

*/\*\**

*\* Constructeur qui initialise le labyrinthe à partir du fichier en paramètre*

*\* @param lab*

*\* @throws FileFormatException*

*\*/*

*public Labyrinthe(File fic) throws FileFormatException*

*/\*\**

*\* Tente de se déplacer dans la case ligne, colonne et de la visiter*

*\* @param ligne*

*\* @param colonne*

*\* @throws ImpossibleMoveException*

*\*/*

*public void move(int ligne, int colonne) throws ImpossibleMoveException*

*/\*\**

*\* deplacement automatique sans deborder*

*\*/*

*public void autoMove()*

*/\*\**

*\* retourne une case de la grille du labyrinthe*

*\* @param lig*

*\* @param col*

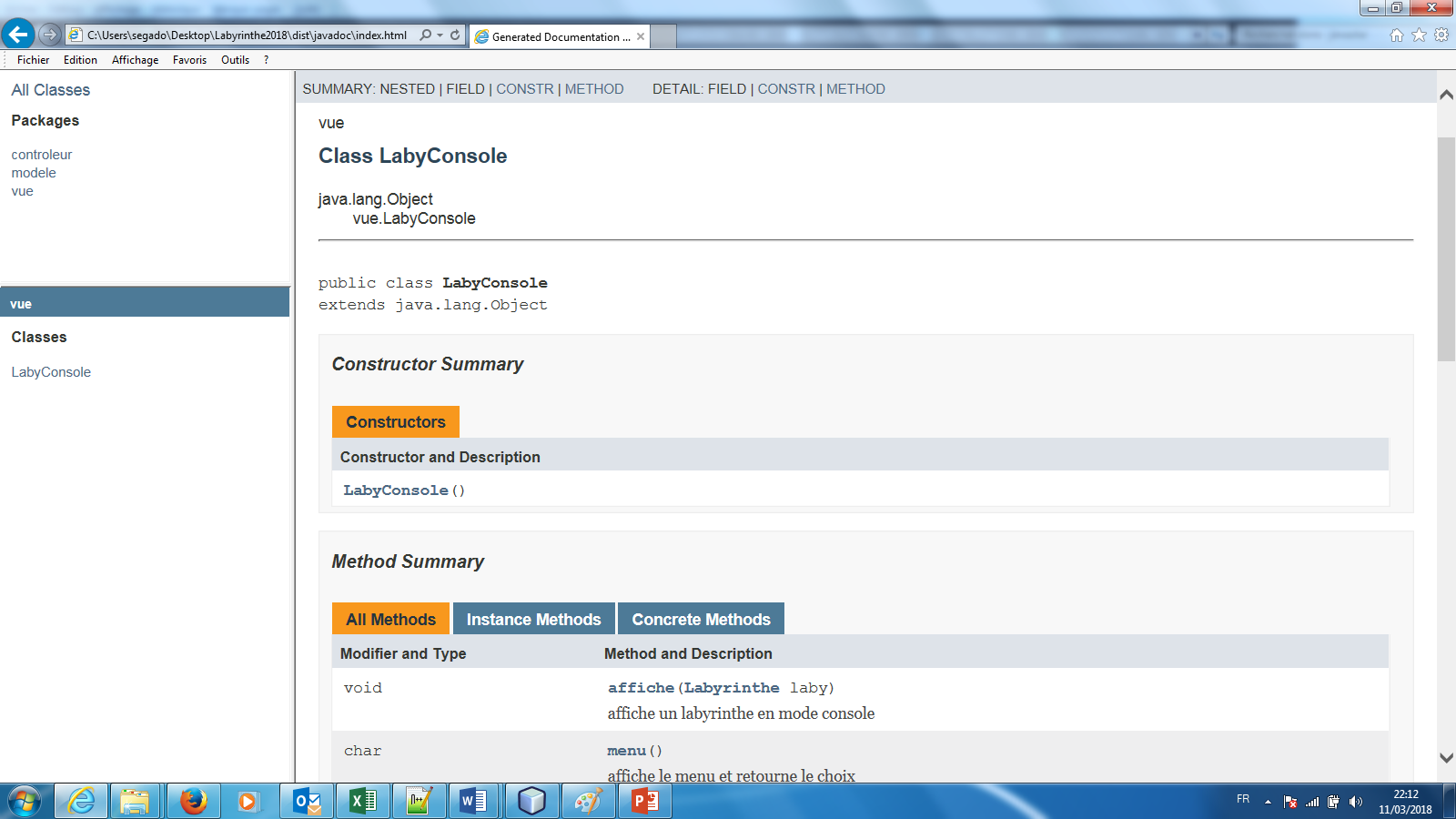
*\* @return*

*\*/*

*public Case getCase(int lig, int col)*

# 3.2) Le package *vue*

Le package *vue* contient la classe *LabyConsole* associée à l’IHM, comme le montre la javadoc ci-dessous :

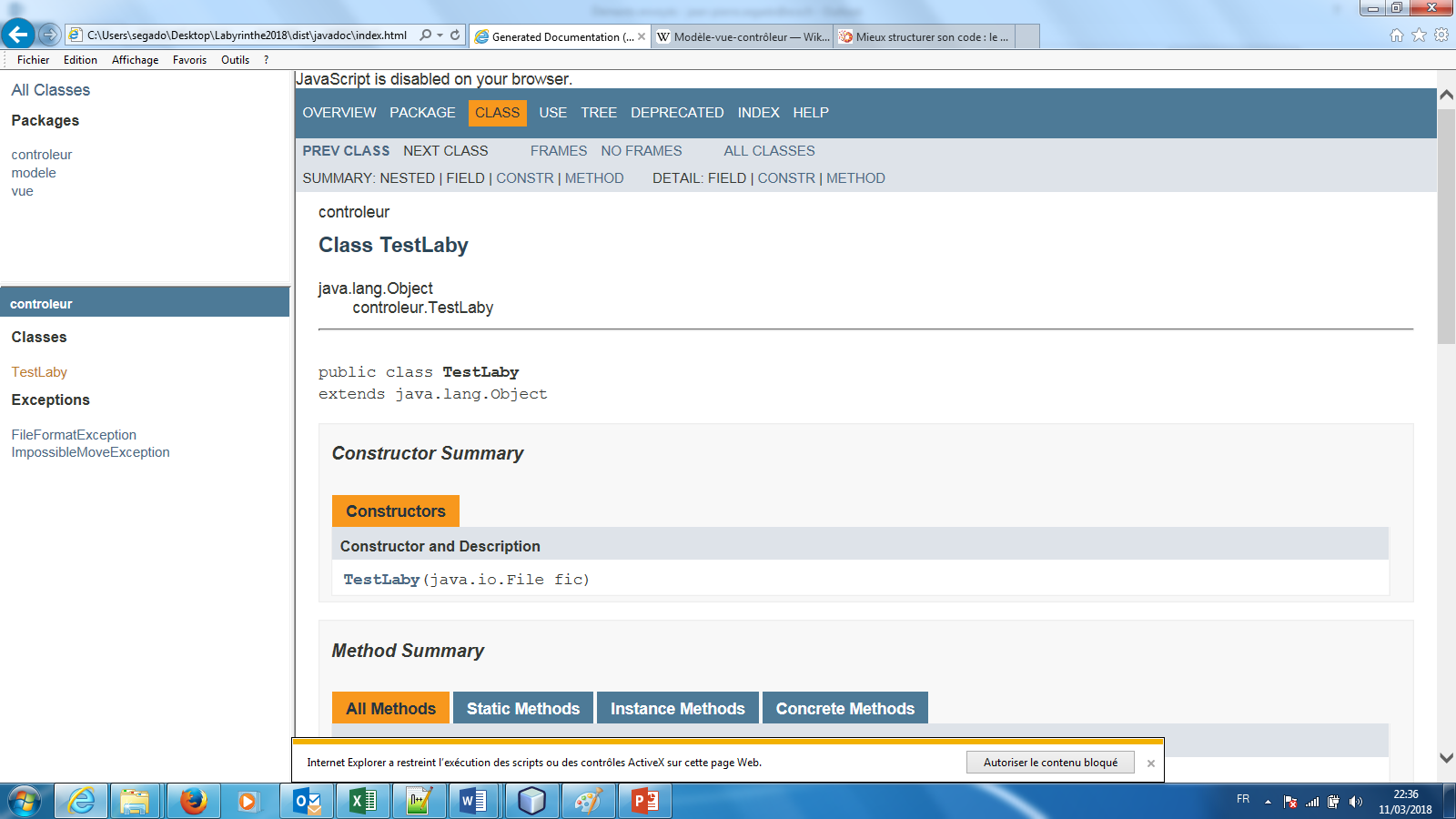


# 3.2) Le package *controleur*

Le *controleur* « permet de faire le lien entre la vue et le modèle lorsqu'une action utilisateur est intervenue sur la vue. C'est cet objet qui aura pour rôle de contrôler les données. »**1**

**1** <https://openclassrooms.com/courses/apprenez-a-programmer-en-java/mieux-structurer-son-code-le-pattern-mvc>

Ce package contient la classe *TestLaby* avec le *main* et les méthodes de déplacement, ainsi que Les classes d’exception *FileFormatException* et *ImpossibleMoveException*



# TRAVAIL A FAIRE :

# Exercice 1 : conception du diagramme de classes

Pour la conception du diagramme de classes, appuyez-vous sur le « Support conception orientée objet » sur la page campus du cours : <http://campus.ece.fr/course/view.php?id=124>.

A l’aide des explications ci-dessus sur le pattern MVC du code et de la javadoc, donnez le diagramme de classes de ce code. N’oubliez pas les niveaux de visibilité pour les attributs et les méthodes (**public** **private** ou **protected**) et les multiplicités des relations inter-classes.

**Pour créer un diagramme de classes avec**[**Draw.io**](https://www.draw.io/)  :

1. Sur DropBox, cliquer sur « Create new diagram »
2. Laisser « Blank Diagram » et faites « Create »
3. Dans le menu à gauche, un sous menu se nomme UML.

Tous les éléments UML sont disponibles dans ce sous-menu. Pour enregistrer la progression « File > Save as ».

Pour exporter en image, pdf, html, etc .. « File > Export as ».

# Exercice 2 : implémentation du labyrinthe en mode graphique

Des extraits utiles de la documentation Javadoc sont fournis en **annexe** : pour plus de détails, consulter la [documentation des API Java *(javadoc*)](http://docs.oracle.com/javase/6/docs/api/) sur la page campus du cours.

Commentez votre code : devant les classes, attributs, constructeurs et méthodes vos commentaires doivent respecter le format Javadoc /\*\* commentaires \*/ pour que ceux-ci apparaissent dans la javadoc générée

* 1. Créer le projet et 3 packages
  2. Dans le premier package, écrire l’interface *Case* avec les prototypes de ses méthodes (voir plus haut)

1. Dans le premier package, écrire la classe *CaseImplementee* qui définit ses attributs **protected**, implémente un constructeur avec en paramètres la position en X et en Y pour initialiser les attributs *posX* et *posY*, ainsi que toutes les méthodes de l’interface *Case*
   1. Dans le premier package, écrire les 2 classes *CaseMur* et *CaseTrou* qui implémentent un constructeur héritant de celui de la classe *CaseImplementee* et spécifie si on peut aller ou non dans la case avec le booléen *vasy*. Pour les objets de la classe *CaseMur*, impossible d’aller dans leur case : voir la méthode *canMoveToCase()* de la classe *Case* et donc une redéfinition de cette méthode est à envisager.
   2. Dans le second package, construire la classe *Labyrinthe* chargée de répertorier ces cases. L’une des 3 approches suivantes à choisir librement peut définir une collection de cases comme attribut :

* Une matrice de cases (*Case* [][]).
* Un *ArrayList* de cases.
* Un *ArrayList* d’*ArrayList* de cases.

La classe *Labyrinthe* devra implémenter les méthodes suivantes en respect de leurs prototypes et des commentaires, sans oublier de définir les attributs privés qui sont précisés plus haut, ni les getters pour y accéder :

Les classes d’exception *FileFormatException* et *ImpossibleMoveException* n’existant pas, vous devez les implémenter.

* 1. Dans le troisième package, écrire une classe avec le *main* avec sa boucle de jeu :
* Introduire un menu avec les options suivantes : déplacement manuel (directions), déplacement automatique intelligent, déplacement automatique aléatoire et sortie du jeu.
* Saisir un nom de fichier, appeler les méthodes nécessaires pour lire le fichier et afficher le labyrinthe.
* Se déplacer du point de départ au point d’arrivée si c’est possible selon l’option du déplacement choisi dans le menu, en affichant au fur et à mesure la position courante.
* En cas d’exceptions *FileFormatException* et *ImpossibleMoveException* afficher des messages d’erreur.

Si nécessaire, implémenter d’autres méthodes dans cette classe comme le déplacement automatique intelligent :

* Par exemple, avec l’algorithme de ces sacrés bons vieux Dijkstra ou BFS (*Breadth First Search* avec une file d’attente) pour avoir le chemin le plus court (nostalgie des ex-ING2 en Théorie des graphes ☺)
* Avec un algorithme récursif si vous voulez traiter tous les chemins possibles (là je sens que vous adorez ☺ ).

# ANNEXE : extraits de la documentation API

**java.io**

**Class File**

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**File**](http://docs.oracle.com/javase/6/docs/api/java/io/File.html#File(java.lang.String))([String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) pathname)  Creates a new File instance by converting the given pathname string into an abstract pathname. |  |

**Class IOException**

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**IOException**](http://docs.oracle.com/javase/6/docs/api/java/io/IOException.html#IOException())()  Constructs an IOException with null as its error detail message. |  |
| [**IOException**](http://docs.oracle.com/javase/6/docs/api/java/io/IOException.html#IOException(java.lang.String))([String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) message)  Constructs an IOException with the specified detail message. |  |

## java.util

## Class Scanner

|  |  |  |
| --- | --- | --- |
| **Constructor Summary** | | |
| [**Scanner**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#Scanner(java.io.File))([File](http://docs.oracle.com/javase/6/docs/api/java/io/File.html) source)  Constructs a new Scanner that produces values scanned from the specified file. | |  |
| [**Scanner**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#Scanner(java.lang.String))([String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) source)  Constructs a new Scanner that produces values scanned from the specified string. | |  |
| **Method Summary** | | |
| void | [**close**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#close())()  Closes this scanner. | |
| boolean | [**hasNext**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#hasNext())()  Returns true if this scanner has another token in its input. | |
| boolean | [**hasNextByte**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#hasNextByte())()  Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the [nextByte()](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextByte()) method. | |
| boolean | [**hasNextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#hasNextInt())()  Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the [nextInt()](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextInt()) method. | |
| boolean | [**hasNextLine**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#hasNextLine())()  Returns true if there is another line in the input of this scanner. | |
| [IOException](http://docs.oracle.com/javase/6/docs/api/java/io/IOException.html) | [**ioException**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#ioException())()  Returns the IOException last thrown by this Scanner's underlying Readable. | |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**next**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#next())()  Finds and returns the next complete token from this scanner. | |
| byte | [**nextByte**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextByte())()  Scans the next token of the input as a byte. | |
| int | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextInt())()  Scans the next token of the input as an int. | |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**nextLine**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextLine())()  Advances this scanner past the current line and returns the input that was skipped. | |

## java.util

**Class ArrayList<E>**

|  |  |  |
| --- | --- | --- |
| **Constructor Summary** | | |
| [**ArrayList**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList())()  Constructs an empty list with an initial capacity of ten. | |  |
| [**ArrayList**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList(java.util.Collection))([Collection](http://docs.oracle.com/javase/6/docs/api/java/util/Collection.html)<? extends [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html)> c)  Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. | |  |
| **Method Summary** | | |
| boolean | [**add**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#add(E))([E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) e)  Appends the specified element to the end of this list. | |
| void | [**add**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#add(int, E))(int index, [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) element)  Inserts the specified element at the specified position in this list. | |
| boolean | [**addAll**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#addAll(java.util.Collection))([Collection](http://docs.oracle.com/javase/6/docs/api/java/util/Collection.html)<? extends [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html)> c)  Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator. | |
| boolean | [**addAll**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#addAll(int, java.util.Collection))(int index, [Collection](http://docs.oracle.com/javase/6/docs/api/java/util/Collection.html)<? extends [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html)> c)  Inserts all of the elements in the specified collection into this list, starting at the specified position. | |
| void | [**clear**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#clear())()  Removes all of the elements from this list. | |
| boolean | [**contains**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#contains(java.lang.Object))([Object](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html) o)  Returns true if this list contains the specified element. | |
| [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**get**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#get(int))(int index)  Returns the element at the specified position in this list. | |
| int | [**indexOf**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#indexOf(java.lang.Object))([Object](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html) o)  Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. | |
| boolean | [**isEmpty**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#isEmpty())()  Returns true if this list contains no elements. | |
| int | [**lastIndexOf**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#lastIndexOf(java.lang.Object))([Object](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html) o)  Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. | |
| [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**remove**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#remove(int))(int index)  Removes the element at the specified position in this list. | |
| boolean | [**remove**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#remove(java.lang.Object))([Object](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html) o)  Removes the first occurrence of the specified element from this list, if it is present. | |
| [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**set**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#set(int, E))(int index, [E](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) element)  Replaces the element at the specified position in this list with the specified element. | |
| int | [**size**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#size())()  Returns the number of elements in this list. | |
| [Object](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html)[] | [**toArray**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#toArray())()  Returns an array containing all of the elements in this list in proper sequence (from first to last element). | |
| |  | | --- | | <T> T[] | | [**toArray**](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. | |

## java.util

**Class Random**

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Random**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#Random())()  Creates a new random number generator. |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| protected int | [**next**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#next(int))(int bits)  Generates the next pseudorandom number. |
| int | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#nextInt())()  Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence. |
| int | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#nextInt(int))(int n)  Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. |