**Academia Java – JAVA**

***Usando la Librería Java***

Tabla de contenido

[1. Usando la librería Java 3](#_Toc312771429)

[1.1 En nuestra última sesión encontramos un bug al juego que estuvimos desarrollando 4](#_Toc312771430)

[1.2 ¿Cómo reparamos esto? 5](#_Toc312771431)

[1.3 Usando el API Java 9](#_Toc312771432)

[1.4 Vamos a reparar el código DotCom 13](#_Toc312771433)

[1.5 Vamos a construir el juego REAL: “Buscar un DotCom” 14](#_Toc312771434)

[1.6 ¿Qué hacemos en el juego DotComBust - y cuando? 17](#_Toc312771435)

[1.7 PrepCode para la clase DotComBust 20](#_Toc312771436)

[1.8 Codigo Real 22](#_Toc312771437)

[1.9 Expresiones Booleanas 25](#_Toc312771438)

[1.10 Clase GameHelper 26](#_Toc312771439)

[Usando Las librerías JAVA 28](#_Toc312771440)

[Soluciones 30](#_Toc312771441)

# Usando la librería Java

Java viene con cientos de clases pre construidas.

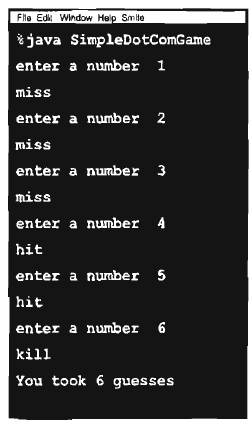
No tenemos que reinventar la rueda debemos buscar en primer lugar en la librería Java antes de hacerlo por nosotros mismos. Dicha librería Java es conocida como **Java API**.



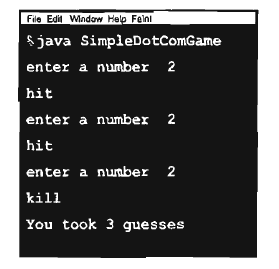
El objetivo de esta sesión es aprender a usarlo.

## 1.1 En nuestra última sesión encontramos un bug al juego que estuvimos desarrollando

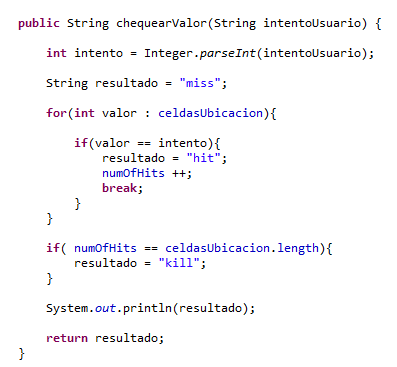
Cuando ingresamos el numero 1,2,3,4,5,6. Todo OK



Pero, cuando ingresabamos 2,2,2. Nos dice que hemos adivinado la palabra, cuando en realidad no es asi.



El problema esta en este metodo:

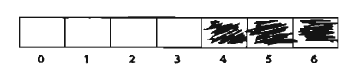


El problema es que se cuenta cada hit que el usuario adivina, aun si la ubicación ya ha sido mencionada.

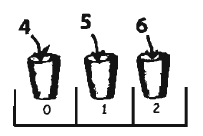
## 1.2 ¿Cómo reparamos esto?

Necesitamos saber si alguna celda ya ha sido adivinada. Recordemos que tenemos 7 celdas y un DotCom ocupa tres celdas consecuitvas en algun lugar de la fila.

La fila virtual con las tres celdas que corresponden al objeto DotCom.

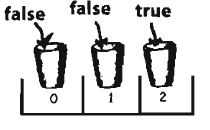


El array que guarda las posiciones del DotCom.



**Opcion 1**

Una opcion para salvar el problema es tener un segundo array que nos guarde los hits. De manera que en cada hit chequeamos si ya se ha adivinado en la celda evaluada.

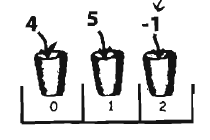


Este array seria de booleans. De esta forma podriamos saber si ya ha sido seleccionada guardando en la celda correspondiente el valor de true.

¿Habra una solucion mejor? \_\_\_\_\_

**Opcion 2**

Nosotros podemos mantener el primer arreglo, pero, cambiamos el valor de la celda adivinada en -1. De esta manera solo chequeamos en un solo arreglo.

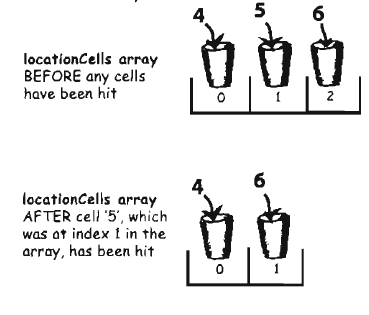


La opcion 2 esta mucho mejor, pero, aun tenemos que recorrer los tres slots (por posicion) en el array, para ver si hemos hecho ‘hit’ ( tiene un valor de -1).

¿Habra alguna solucion mejor? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Opcion 3**

Borramos cada celda adivinada y trabajamos con el nuevo array resultante.



|  |  |
| --- | --- |
| http://1.bp.blogspot.com/-AHrFEZDBbWs/TdPxajPWbeI/AAAAAAAAE8s/nHhzlcp-7YQ/s1600/pensando2.jpg | La opcion 3 esta mucho mejor si el array se pudiera generar sin hacer un array demas, sin tener que copiar los valores restantes y reasignar las referencias. **¿Existira algo asi?** |

El prepcode cambiaria de su forma original a la siguiente forma:

**MÉTODO**: *String checkYourself(String userGuess)*

**OBTENER** la suposición del usuario como un parámetro *String*.

**CONVERTIR** la suposición del usuario a un *int*.

**REPETIR** por cada localización de las celdas en el *array* de *int*

**//COMPARAR** la suposición del usuario con la localización de las celda

**SI** la suposición es correcta

**INCREMENTAR** el número de hits

**//AVERIGUAR** si fue la última celda

**SI** la cantidad de hits es 3

**RETORNA** “kill” como resultado

**CASO** **CONTRARIO** si no fue un kill

**RETORNA** “hit”

**FIN** **SI**

**CASO** **CONTRARIO** si la suposición no es correcta

**RETORNA** “miss”

**FIN SI**

**FIN** **REPETIR**

**FIN** **MÉTODO**

A esta:

**MÉTODO**: *String checkYourself(String userGuess)*

**OBTENER** la suposición del usuario como un parámetro *String*.

**CONVERTIR** la suposición del usuario a un *int*.

**REPETIR** con cada una de las celdas restantes

**//COMPARAR** la suposición del usuario con la localización de las celda

**SI** la suposición es correcta

**REMOVER**  celda del arreglo

**//AVERIGUAR** si fue la última celda

**SI** el array esta vacio

**RETORNA** “kill” como resultado

**CASO** **CONTRARIO** si no fue un kill

**RETORNA** “hit”

**FIN** **SI**

**CASO** **CONTRARIO** si la suposición no es correcta

**RETORNA** “miss”

**FIN SI**

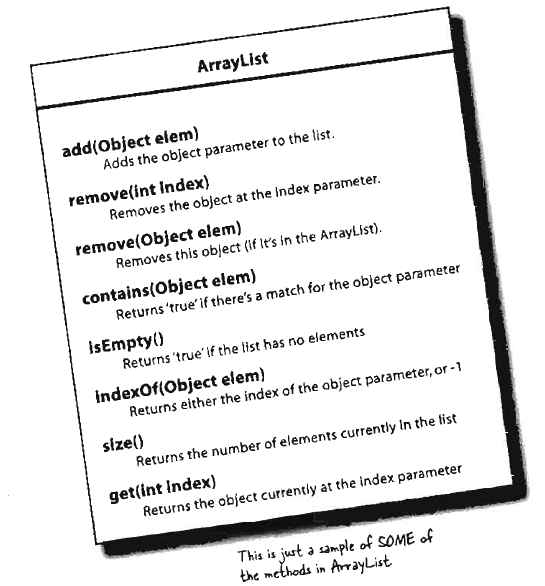
**FIN** **REPETIR**

**FIN** **MÉTODO**

## 1.3 Usando el API Java

# 

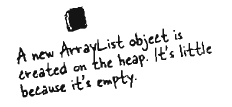
Lo que buscamos si existe. Se llama ArrayList. No es un array, pero, nos brinda la funcionalidad buscada. Es una clase que esta en el API Java.



**Que podemos hacer con un ArrayList**

1. Hacer uno

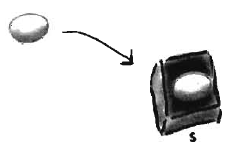
ArrayList<Egg> myList = new ArrayList<Egg>();



1. Ponemos algo en este

Egg s = new Egg();

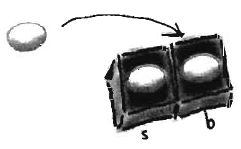
myList.**add**(s);



1. Ponemos otro huevo en la lista

Egg b = new Egg();

myList.add(b);



1. Cuantas cosas tenemos en la lista

int tamanno = myList.**size()**; //2

1. Verificar si un elemento existe en la lista

boolean esta = myList.**contains**(s); //si lo contiene al objeto s, retorna true.

1. Encontrar la posicion o indice donde esta ubicado el objeto

int idx = myList.**indexOf**(b); //devuelve el indice donde esta ubicado. En este caso 1

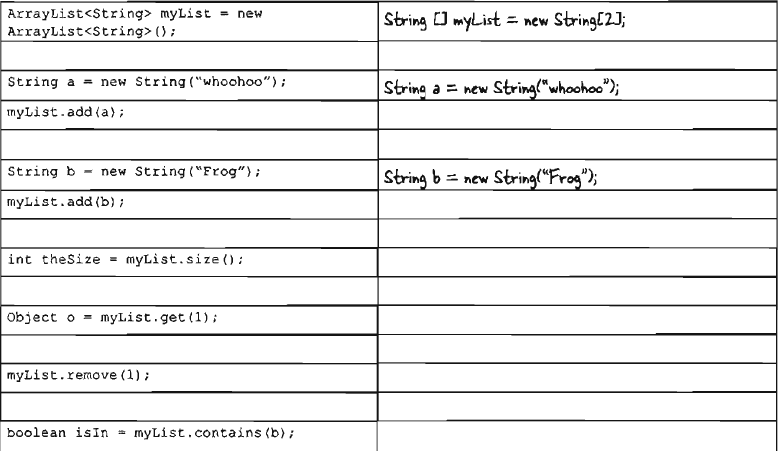
1. Ver si la lista esta vacia.

boolean vacio = myList.**isEmpty()**;

1. Remover un elemento de la lista.

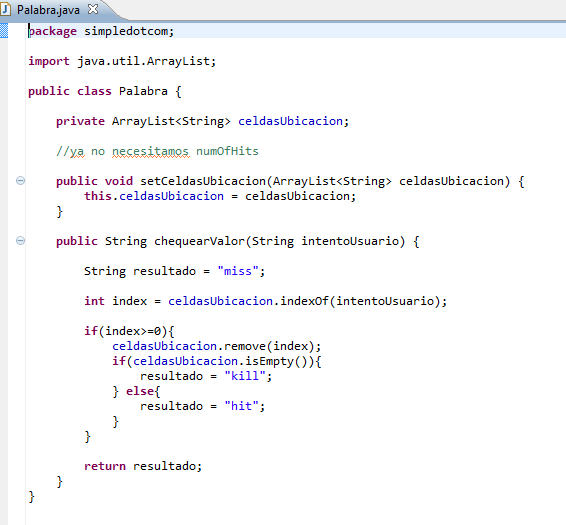
myList.**remove**(s);

|  |  |
| --- | --- |
|  | **Dada la siguiente tabla, cual seria el codigo correspondiente en arrays.** |

****

Solucion: Al final de la unidad

## 1.4 Vamos a reparar el código DotCom



## 1.5 Vamos a construir el juego REAL: “Buscar un DotCom”

El juego a construir REAL es una grilla de 7x7 y tres palabras DotCom. Cada DotCom tendrá tres celdas.

|  |  |
| --- | --- |
|  |  |

**¿Qué necesita cambiar?**

Tenemos tres clases que necesitan cambiar: Palabra (DotCom), la clase Juego (DotComBust) y la clase Helper.

1. La clase DotCom
   * Anadir una variable name

La cual guardara el nombre de la palabra DotCom (“pets.com”, “peru.com”, “groupon.com”, etc). De manera que cada DotCom puede decir su nombre cuando se ha adivinado su ubicación.

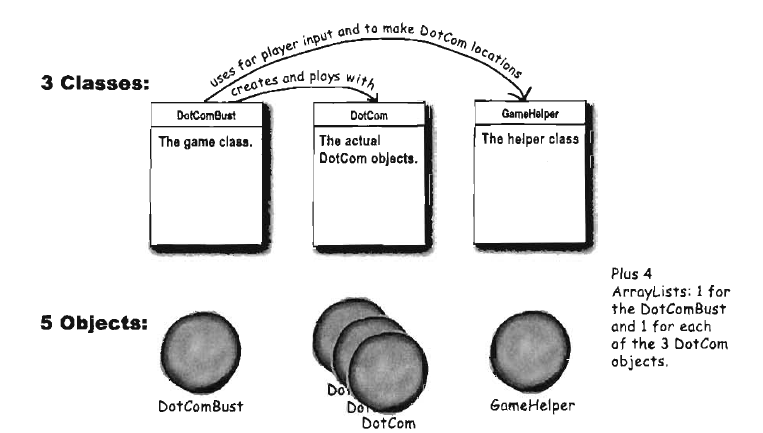
1. La clase DotComBust (el juego)
   * Crea las tres palabras DotComs en lugar de uno.
   * Le da a cada DotCom un nombre.

Se llama a un método setter en cada instancia DotCom, de manera que cada DotCom tenga un nombre acorde a su variable de instancia name.

* + Se coloca el DotCom en la grilla en lugar de una simple fila, y se hace esto para los tres DotComs.

Este paso es ahora más complejo que antes. Pero estemos tranquilos porque nos darán el algoritmo para colocar de manera aleatoria la palabra DotCom en la clase GameHelper.

* + Chequear que cada usuario adivina los tres DotComs y no solo uno
  + Mantenernos en el juego hasta que no haya mas DotComs por adivinar.
  + Salir del main. Cuando termine el juego.



El plus 4: 1 ArrayList para el DotComBust y 1 por cada uno de los tres objetos DotCom.

## 1.6 ¿Qué hacemos en el juego DotComBust - y cuando?

|  |  |
| --- | --- |
|  | El metodo main() en la clase DotComBust instancia el objeto DotCombBust que es el juego propiamente dicho. |

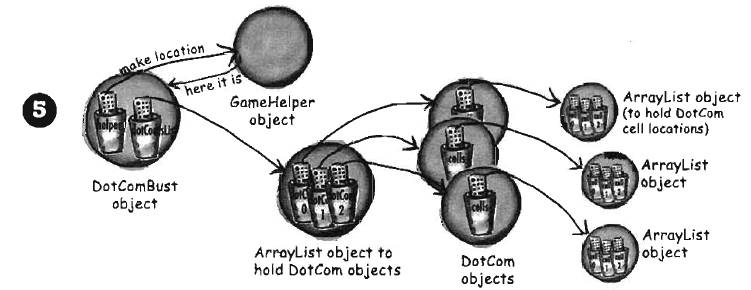
|  |  |
| --- | --- |
|  | El juego (DotComBust) instancia un objeto GameHelper que le ayudara a hacer su trabajo. |

|  |  |
| --- | --- |
|  | El objeto DotComBust instancia un ArrayList que guardara los 3 objetos DotCom. |

|  |  |
| --- | --- |
|  | El objeto DotComBust crea tres objetos DotCom ( y pone ello en el ArrayList) |

El objeto DotComBust consulta al objeto Helper por una ubicación para un DotCom (lo haremos tres veces para poder ubicar cada uno de los DotCom).

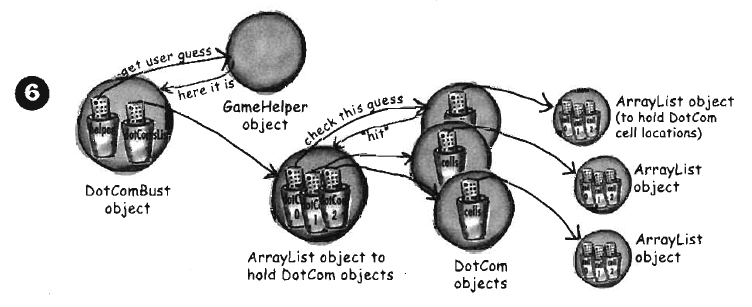
El objeto DotComBust dara a cada uno de los objetos DotCom una ubicación (obtenido del objeto helper) como “A2”, “B2”, etc. Cada objeto DotCom tiene a su vez su ArrayList para guardar las tres posiciones en la grilla.



Esta parte es más compleja, pero, es como funciona:

El objeto DotComBust consulta al helper por la entrada de un usuario.

El objeto DotComBust busca en la lista de DotComs y pregunta a cada uno si el valor ingresado por el usuario es acertado. El DotCom cheque su arrayList de ubicaciones y retorna como resultado “hit”, “miss”, etc.



El juego continua hasta que el usuario adivina cada DotCom.

## PrepCode para la clase DotComBust

La clase DotComBust tiene tres trabajos principales: configurar el juego, jugar el juego hasta que los DotComs han sido adivinados, y finalizar el juego. Haremos la funcionalidad del juego en dos métodos, para mantener la granularidad lo mas pequeña posible. Métodos pequeños son más fáciles de testear, depurar y modificar.

**Declaraciones de Variables**

**DECLARA** una instancia de GameHelper, y lo denominaremos helper.

**DECLARA** una instancia de ArrayList para guardar la lista de DotComs (inicialmente tres). Lo llamaremos dotComsList.

**DECLARA** una variable int para guardar el numero de intentos del usuario (de esta forma al final se le dara al usuario su score). Lo denominaremos numOfGuesses y se configurara inicialmente a cero.

**Declaraciones de Métodos**

**DECLARA** un método setupGame() para crear e iniciar los objetos DotCom con nombres y ubicaciones. Muestra unas breves instrucciones al usuario.

**DECLARA** un método startPlaying() que consulta al jugador por su alternativa y llama al método checkUserGuess() hasta que todos los objetos DotCom han sido removidos del juego.

**DECLARA** un método checkUserGuess() que busca, a través, de los objetos DotComs restantes y llamada por cada objeto DotCom a su método checkYourself().

**DECLARA** un método finishGame() que imprime un mensaje acerca de la performance del usuario, basado en el numero de intentos que le tomo para adivinar los tres DotCom.

**Implementaciones de Métodos**

**METODO: void setUpGame()**

//haremos los tres objetos DotCom y le damos un nombre a cada uno

**CREAR** los tres objetos DotCom

**ESTABLECER** un nombre por cada DotCom

**AÑADIR** los DotComs a la lista dotComList (ArrayList)

**REPETIR** con cada uno de los objetos DotCom en el array dotComsList

**LLAMAR** el método placeDotCom() en el objeto helper, para obtener una ubicación aleatoria para este DotCom (tres celdas, vertical o horizontalmente alineadas, en una grilla 7x7)

**ESTABLECER** la ubicación para cada DotCom basado en el resultado de la llamada a placeDotCom()

**FIN REPETIR**

**FIN METODO**

**METODO: void startPlaying()**

**REPETIR** mientras algún DotComs exista

**OBTENER** entrada del usuario llamando al metodo helper getUserInput()

**EVALUAR** la entrada del usuario con el metodo checkUserGuess()

**FIN REPETIR**

**FIN METODO**

**METODO: void checkUserGuess(String userGuess)**

//encontremos si hemos hecho hit o kill en algun DotCom

**INCREMENTA** el numero de intentos en la variable numOfGuesses

**ESTABLECE** el valor de variable result a “miss”, asumiendo que el usuario falla en su intento.

**REPETIR** con cada uno de los objetos DotObjects en el array dotComList

**EVALUAR** la adivinanza del usuario llamado al metodo checkYourself() de cada objeto DotCom

**ESTABLECER** la variable result a “hit” o “kill” según sea apropiado

**SI** el resultado es “kill”, **REMUEVE** el DotCom de la lista dotComsList

**FIN REPETIR**

**MOSTRAR** el valor de result a el usuario

**FIN METODO**

**METODO: void finishGame()**

**MOSTRAR** un mensaje generio “game over”, luego:

**SI** numero de intentos del usuario es pequeño

**MOSTRAR** un mensaje de felicitacion

**CASO CONTRARIO**

**MOSTRAR** un mensaje de fail

**FIN SI**

**FIN METODO**

## Codigo Real

**DotComBust**

|  |
| --- |
| **import java.util . \*;**  **public class DotComBust{**  **GameHelper helper new GameHelper() ;**  **private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();**  **private int numOfGuesses = 0 ;**  **private void setUpGame () (**  **//first make some dot corns and give them locations**  **DotCom one = new DotCom();**  **one .setName(“Pets.com");**  **DotCom two = new DotCom();**  **two.setName("eToys.com”) ;**  **DotCom three = new DotCom();**  **three .setName{ "Go2.com");**  **dotComsList.add(one);**  **dotComsList.add(two);**  **dotComsList.add(three);**    **for (DotCom pal : dotComsList) (**  **ArrayList<String> newLocation = helper .placeDotCom(3);**  **pal.setLocationcells(newLocation);**  **}**  **}**    **private void startPlaying () (**  **while(!dotComsList.isEmpty() (**  **String userGuess = helper.getUserInput("Enter a guess");**  **checkUserGuess(userGuess);**  **}**  **finishGame();**  **}**    **private void checkUserGuess (String userGuess) {**  **numOfGuesses++;**  **String result = "miss" ;**  **for (DotCom dotComToTest : dotComsList) {**  **result = dotComToTest.checkYourself(userGuess);**  **if (result.equals("hit") {**  **break;**  **}**  **if (result.equals("kill")) {**  **dotComsList.remove(dotComToTest);**  **break;**  **}**  **}**  **System.out.println(result);**  **}**    **private void finishGame(){**  **System.out.println("All Dot Corns are dead! Your stock i s now worthless.");**  **if (numOfGuesses <= 18) {**  **System.out.println(" It only took you" + numOfGues ses + " guesses.") ;**  **System.out.println(" You got out before your op tio n s sank.");**  **}else {**  **System.out.println("Took you long enough. " + numOfGuesses + " guesse s .");**  **System.out.println("Fish are dancing with your op tion s . " ) ;**  **}**  **}**      **public static void main(String[] args) {**  **DotComBust game = new DotComBust();**  **game.setUpGame();**  **game.startPlaying();**  **}**    **}** |

**Clase DotCom**

|  |
| --- |
| **import java .util.\*;**  **public class DotCom {**  **private ArrayList<String> locationcells;**  **private String name;**    **public void setLocationCells(ArrayList<String> loc) {**  **locationCells = loc;**  **}**    **public void setName (String n) {**    **name = n;**  **}**    **public String checkYourself(String userlnput) {**  **String result = "miss";**  **int index = locationCells .indexOf(userlnput);**    **if (index >= 0) {**  **locationCells.remove(index);**  **if (locationCells. isEmpty ()) {**  **result = "kill";**  **System.out.println("Ouch! You sunk " + name + " ( ");**    **}else{**  **result = "hit ";**  **}**  **}**  **return result;**    **}**  **}** |

## Expresiones Booleanas

En los códigos mostrados hemos observado bastante el uso de expresiones booleanas para nuestros loops, if tests, y ha sido muy simple su uso. Vamos a usar expresiones booleanas mas avanzadas en el código de la clase Helper.

**Operadores ‘And’ y Or’ ( &&, ||)**

“Si el rango de precio de una camara esta entre $300 y $400 entonces escogemos X”

**if( price >=300 && price < 400) {**

**camara = “x”;**

**}**

Vamos a decir que hay más de 10 marcas de cámaras disponibles, entonces aplicaremos una lógica como esta:

**if( brand.equals(“A”) || brand.equals(“B”)) {**

**// hacer algo con A o con B**

**}**

Expresiones Booleanas pueden ser muy grandes y complicadas:

**if ((zoomType.equals("optical") &&**

**(zoomDegree >= 3 && zoomDegree <= )) II**

**(zoomType.equals("digital") &&**

**(zoomDegree >= 5 && zoomDegree <= 12))) {**

***//hacer algo***

**}**

**RECOMENDACION: Usar paréntesis.**

**Not equals (!= y !)**

**if (model != 2000) {**

***// hacer algo***

**}**

O para comparar String u objetos

**if (!brand.equals(“X”)) {**

***//hacer algo***

**}**

**& , |**

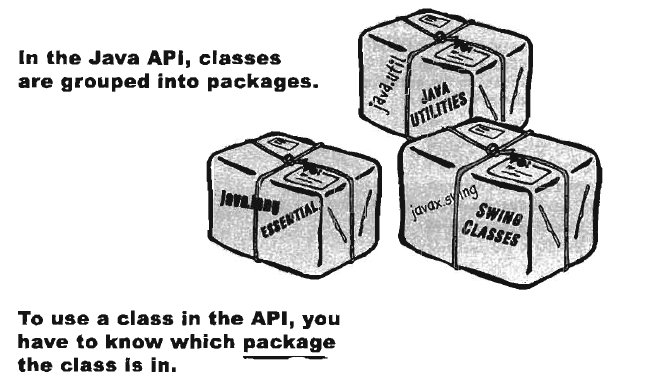
Son para operar bits

## Clase GameHelper

Esta es la clase GameHelper que hace uso de muchas expresiones booleanas.

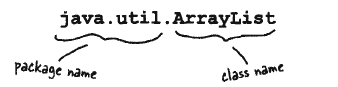
|  |
| --- |
| import java.io.\*;  import java.util.\*;  public class GameHelper {  private static final String alphabet = "abcdefg";  private int gridLength = 7:  private int gridSize = 49;  private int [] grid = new int[gridSize);  private int comCount = 0;  public String getUserlnput(String prompt){  String inputLine = null;  System.out.print(prompt +" ");  try {  BufferedReader is = new BufferedReader(  new InputStreamReader(System.in));  inputLine = is.readLine();  if (inputLine.length null () == 0) return;  } catch (IOException e) {  System.out.println("IOException : " + e);  }  return inputLine .toLowerCase();  }    public ArrayList<String> placeDotCom(int comSize) {  ArrayList<String> alphaCells = new ArrayList<String>();  String [] a1phacoords = new String [comSize];  String temp = null;  int [] coords = new int[comSize];  int attempts = 0;  boolean success = false;  int location = 0;    comCount++;  int incr = 1;  if ((cornCount % 2) == 1) {  incr = gridLength;  }    while ( !success & attempts++ < 200 ) {  location = (int) (Math.random() \* gridSize);  //System.out.print( " t ry" + location);  int x = O;  success = true;  while (success && x < comSize) {  if (grid[locacion] == 0) {  coords[x++] = location;  location += incr;  if (location>= gridSize){  success = false;  }  if (x>O && (location % gridLength ==0)) {  success = false;  }  } else {  //System.out.print(" used " + location);  success = false;  }  }        }    int x = 0;  int row = 0;  int column = 0;  //System.out.println( "\ n " ) ;  while (x < c omSize) {  grid[coords[x]] = 1;  row = (int) (coords[x] / gridLength);  column = coords[x] % gridLength;  temp = String.valueOf(alphabet.charAt(column));  alphaCells.add(temp.concat(Integer.toString(row)));  x++;    //System.out.print(" coord "+x +" = "+ alphaCells.get(x-l));  }  return alphaCells;    }  } |

# Usando Las librerías JAVA



Nosotros ya hemos estado usando un paquete: **java.lang.\***

**Tenemos que saber el nombre completo de una clase que deseamos usar**

****

**Tenemos que decirle a Java que deseamos usar ArrayList. Tenemos dos opciones:**

1. Usando import



1. Usando el nombre completo



|  |  |
| --- | --- |
| * ArrayList es una clase en el API Java * Para poner algo en un ArrayList usamos add() * Para remover algo del ArrayList usamos remove() * Para encontrar algo (si esta) en un ArrayList usamos indexOf() * Para saber si un ArrayList esta vacio usamos isEmpty() * Para obtener el tamanno de elementos en un ArrayList usamos el metodo size() * Para obtener la longitud en un array usamos la variable length * Un ArrayList se redimensiona dinamicamente ya sea añadiendo, o eliminando objetos * Las clases tienen un nombre completo, el cual es una combinacion del nombre de paquete y el nombre de clase. La clase ArrayList es realmente java.util.ArrayList * Para usar una clase que no esta en java.lang tenemos que decirle a java el nombre completo ya sea usando una sentencia import o a la hora de declarar usar el nombre completo. | C:\Users\ingas\Desktop\Atencion2.png |

# Soluciones

1. Comparando ArrayList con un array

