Get to Know the Using the Java Library

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교
HANKYONG NATIONAL UNIV.

The cliff-hanger. A bug

How it's supposed to look

숫자 1,2,3,4,5,6을 입력하여 어떻게 실행되는가 확인
=> Lookin' good.

A complete game interaction

```
File Edit Window Help Smile
%java SimpleDotComGame
enter a number  1
miss
enter a number  2
miss
enter a number  3
miss
enter a number  4
hit
enter a number  5
hit
enter a number  6
kill
You took 6 guesses
```

A different game interaction (yikes)

How the bug looks

2, 2, 2 를 입력했을 때 무슨 일이 일어날까?

현재 버전에서는 이미 히트한 숫자를 계속 입력해도 누적 히트 수를 증가시킨다!!

```
File Edit Window Help Faint
%java SimpleDotComGame
enter a number  2
hit
enter a number  2
hit
enter a number  2
kill
You took 3 guesses
```

```
public String checkYourself(String stringGuess) {

    int guess = Integer.parseInt(stringGuess);

    String result = "miss";

    for (int cell : locationCells) {

        if (guess == cell) {

            result = "hit";

            numOfHits++;

            break;

        } // end if

    } // end for

    if (numOfHits == locationCells.length) {

        result = "kill";

    } // end if

    System.out.println(result);

    return result;

} // end method
```

Convert the String to an int.

Make a variable to hold the result we'll return. Put "miss" in as the default (i.e. we assume a "miss").

Repeat with each thing in the array.

Compare the user guess to this element (cell), in the array.

we got a hit!

Get out of the loop, no need to test the other cells.

We're out of the loop, but let's see if we're now 'dead' (hit 3 times) and change the result String to "kill".

Display the result for the user ("miss", unless it was changed to "hit" or "kill").

Return the result back to the calling method.

Here's where it goes wrong. We counted a hit every time the user guessed a cell location, *even if that location had already been hit!*

We need a way to know that when a user makes a hit, he hasn't previously hit that cell ... then

이미 히트했다면
카운트하지 않는
방법이 필요하다!!

How do we fix it ?

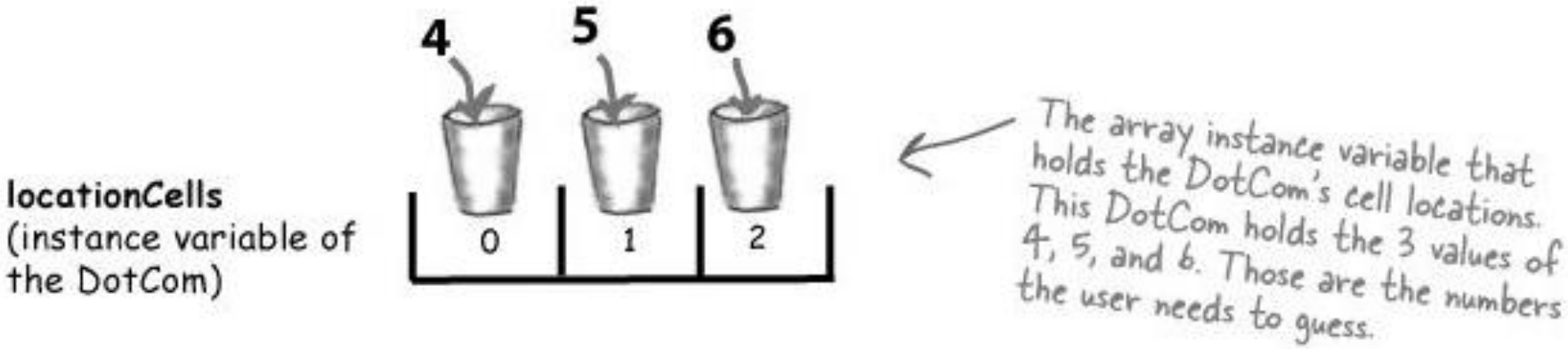
셀이 이미 히트되었는지 알 방법이 필요하다.

몇 가지 가능성을 살펴보기 전에 먼저 지금까지 파악한 내용을 정리해보자:

7개의 셀을 가진 가상 행에서 그 행의 어딘가에 3개의 연속된 셀을 차지하는 닷컴이 있을 것이다.

아래 가상 행은 닷컴이 셀 위치 4,5,6에 배치되었음을 보여준다.

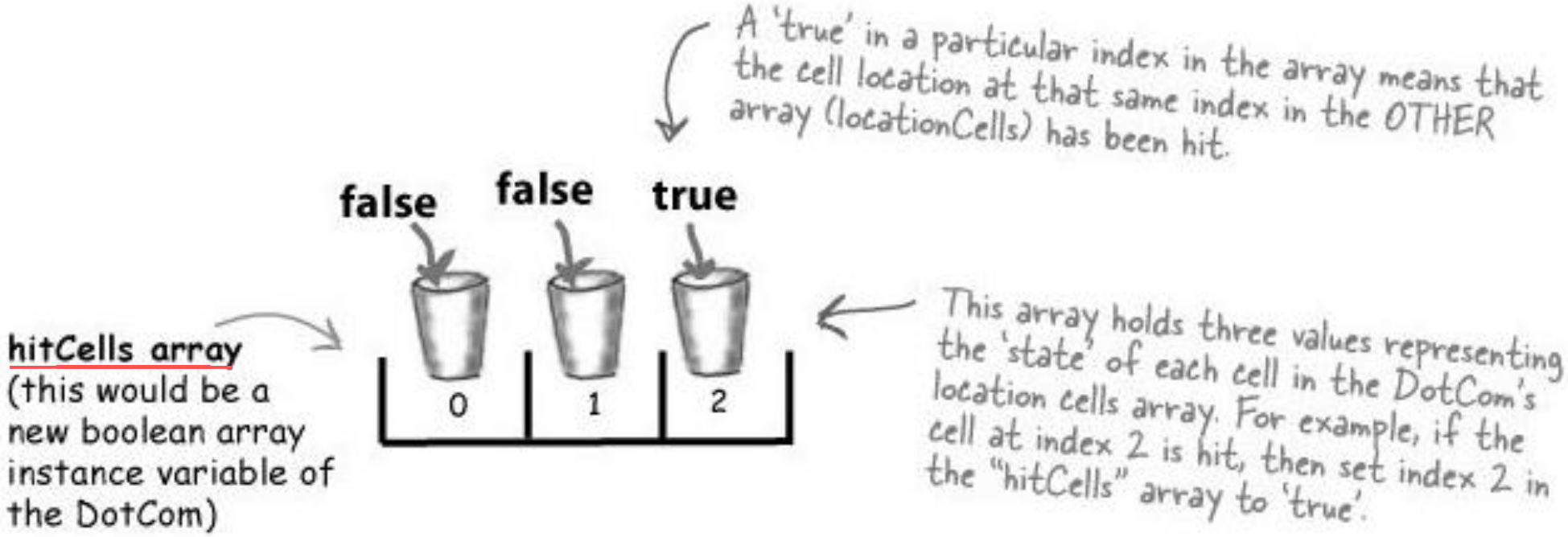
DotCom은 닷컴 객체의 셀 위치를 저장하는 인스턴스 변수(int 배열)를 갖는다.



Option one

두 번째 배열(**'hitCells'**)을 새로 만들어서 사용자가 히트할 때마다 **'hitCells'** 배열에 히트 상태를 저장한다.

다음 단계로 사용자가 히트할 때마다 그 셀이 전에 히트되었는지 확인하기 위해 **'hitCells'** 배열을 체크한다.



Option one is too clunky

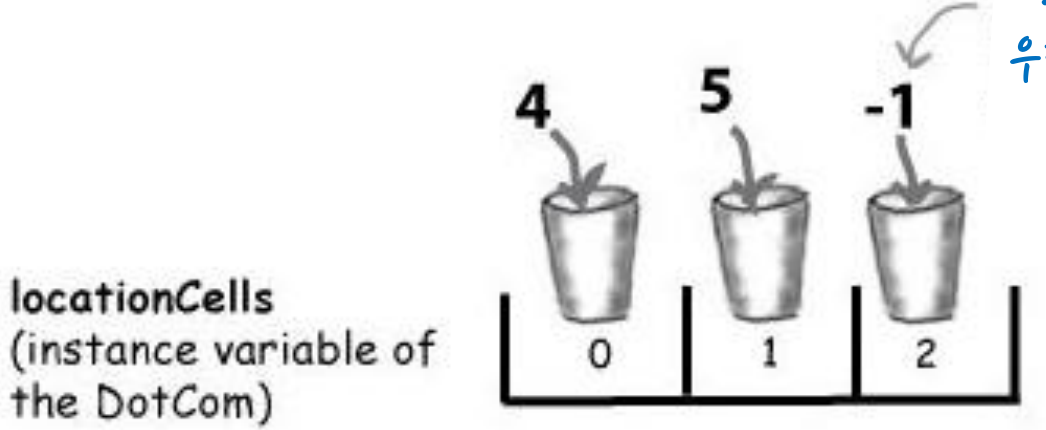
사용자가 히트할 때마다 두 번째 배열 ('**hitCells**' 배열)의 상태를 변경해야 ...

그리고 그 셀이 이미 히트되었는 지를 알아보기 위해 먼저 '**hitCells**' 배열을 체크해야만 한다.

⇒ 실행은 된다! 그러나 좀 비효율적이다...

Option two

셀이 이미 히트되었다면 그 셀 위치에 -1을 넣는다.
우리는 오로지 양의 정수만 찾는다.

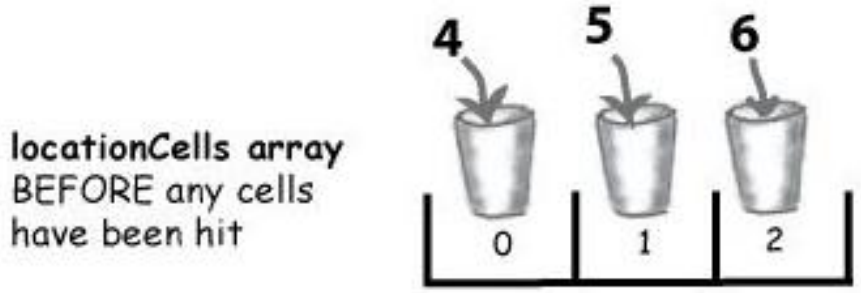


Option two is a little better, . . .

Option Two 문제점:

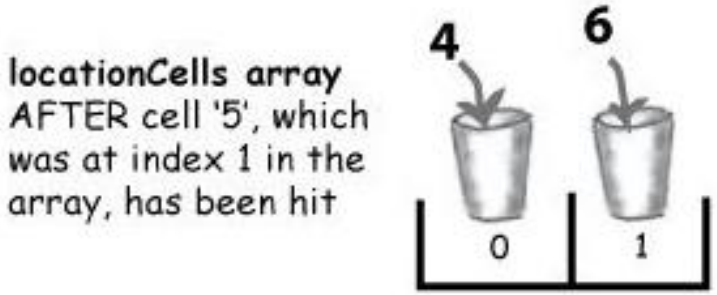
셀이 히트되어서 하나 이상의 셀이 이미 **invalid** 되었음에도 불구하고 여전히 배열의 세 개의 셀 위치에 대해 반복해서 체크해야만 한다.

Option three



셀이 히트되면 각 셀 위치를 제거한다

⇒ 배열의 크기가 줄어든다.



배열을 더 이상 줄일 수 없을 때까지 **new** 배열을 만들어야 한다

⇒ 이전 배열보다 더 작은 새로운 배열로 나머지 셀들이 복사된다.

Option three

The original prepcode for part of the checkYourself() method:

```
REPEAT with each of the location cells in the int array
    // COMPARE the user guess to the location cell
    IF the user guess matches
        INCREMENT the number of hits
        // FIND OUT if it was the last location cell:
        IF number of hits is 3, RETURN "kill"
        ELSE  it was not a kill, so RETURN "hit"
        END IF
    ELSE  user guess did not match, so RETURN "miss"
    END IF
END REPEAT
```

Life would be good if only we could change it to:

```
REPEAT with each of the remaining location cells
    // COMPARE the user guess to the location cell
    IF the user guess matches
        REMOVE this cell from the array
        // FIND OUT if it was the last location cell:
        IF the array is now empty, RETURN "kill"
        ELSE  it was not a kill, so RETURN "hit"
        END IF
    ELSE user guess did not match, so RETURN "miss"
    END IF
END REPEAT
```

항목을 제거하면 크기가 줄어드는
배열이 과연 있을까??

Wake up and smell the library

ArrayList
add(Object elem)
Adds the object parameter to the list.
remove(int index)
Removes the object at the index parameter.
remove(Object elem)
Removes this object (if it's in the ArrayList).
contains(Object elem)
Returns 'true' if there's a match for the object parameter
isEmpty()
Returns 'true' if the list has no elements
indexOf(Object elem)
Returns either the index of the object parameter, or -1
size()
Returns the number of elements currently in the list
get(int index)
Returns the object currently at the index parameter

마치 매직처럼 그런 것이 있다.

그런데 그것은 **Array**가 아니다.

그것은 바로 **ArrayList**이다.

⇒ 이 클래스는 코어 자바 라이브러리(API)에 포함되어 있다.

https://docs.oracle.com/javase/8/docs/api/

Some things you can do with ArrayList

① Make one

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

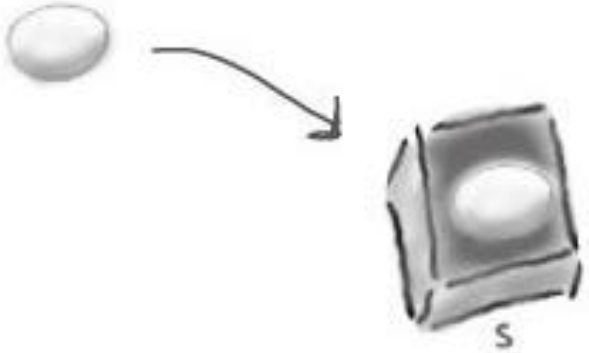
Don't worry about this new <Egg> angle-bracket syntax right now; it just means "make this a list of Egg objects".

A new ArrayList object is created on the heap. It's little because it's empty.

② Put something in it

```
Egg s = new Egg();

myList.add(s);
```

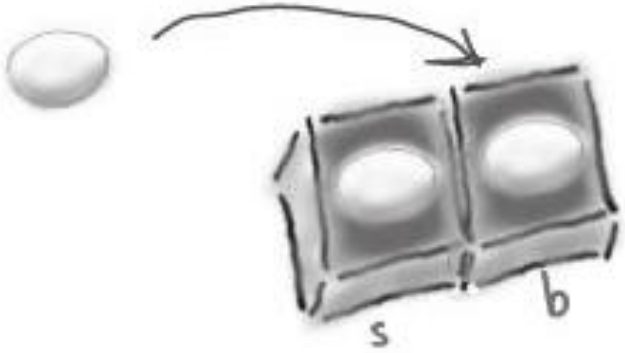


Now the ArrayList grows a "box" to hold the Egg object.

③ Put another thing in it

```
Egg b = new Egg();

myList.add(b);
```



The ArrayList grows again to hold the second Egg object.

④ Find out how many things are in it

```
int theSize = myList.size();
```

The ArrayList is holding 2 objects so the size() method returns 2

⑤ Find out if it contains something

```
boolean isIn = myList.contains(s);
```

The ArrayList DOES contain the Egg object referenced by 's', so contains() returns true

⑥ Find out where something is (i.e. its index)

```
int idx = myList.indexOf(b);
```

ArrayList is zero-based (means first index is 0) and since the object referenced by 'b' was the second thing in the list, indexOf() returns 1

⑦ Find out if it's empty

```
boolean empty = myList.isEmpty();
```

it's definitely NOT empty, so isEmpty() returns false

⑧ Remove something from it

```
myList.remove(s);
```

Hey look — it shrank!

Comparing ArrayList to a regular array

ArrayList	regular array
`ArrayList<String> myList = new ArrayList<String>();`	`String [] myList = new String[2];`
`String a = new String("whoohoo");`	`String a = new String("whoohoo");`
`myList.add(a);`	`myList[0] = a;`
`String b = new String("Frog");`	`String b = new String("Frog");`
`myList.add(b);`	`myList[1] = b;`
`int theSize = myList.size();`	`int theSize = myList.length;`
`Object o = myList.get(1);`	`String o = myList[1];`
`myList.remove(1);`	`myList[1] = null;`

`boolean isIn = myList.contains(b);`	`boolean isIn = false;`
	`for (String item : myList) {`
	`if (b.equals(item)) {`
	`isIn = true;`
	`break;`
	`}`
	`}`

Here's where it starts to look really different...

Comparing ArrayList to a Regular Array

1. **Regular Array**는 생성될 때 미리 배열의 크기를 알아야만 한다.
   그러나 **ArrayList**는 그 크기를 알 필요가 없다.
   (객체가 추가되고 제거됨에 따라 커지고 줄어들 수 있다)

```
new String[2]
```

Needs a size.

```
new ArrayList<String>()
```

No size required (although you can give it a size if you want to).

2. 객체를 **Regular Array**에 넣기 위해서는 배열의 특정 위치를 지정해줘야 한다.
   ArrayList에서는 **add(anInt, anObject)** 또는 **add(anObject)**를 사용하여 인덱스를 지정할 수 있다.

```
myList[1] = b;
```

Needs an index.

```
myList.add(b);
```

No index.

3. **Array**는 오로지 배열을 위해서만 사용되는 특별한 문법을 사용한다.

```
myList[1]
```

The array brackets [ ] are special syntax used only for arrays.

4. **ArrayList**는 Java 5.0부터 파라메터화 되었다(parameterized).

```
ArrayList<String>
```

The <String> in angle brackets is a "type parameter". ArrayList<String> means simply "a list of Strings", as opposed to ArrayList<Dog> which means, "a list of Dogs".

닷컴 코드를 고쳐보자

This is how the buggy version looks:

```
public class DotCom {

   int[] locationCells;
   int numOfHits = 0;

   public void setLocationCells(int[] locs) {
      locationCells = locs;
   }

   public String checkYourself(String stringGuess) {
      int guess = Integer.parseInt(stringGuess);
      String result = "miss";

      for (int cell : locationCells) {
         if (guess == cell) {

            result = "hit";
            numOfHits++;

            break;
         }
      } // out of the loop

      if (numOfHits == locationCells.length) {
          result = "kill";
      }
      System.out.println(result);
         return result;
    } // close method
} // close class
```

We've renamed the class DotCom now (instead of SimpleDotCom), for the new advanced version, but this is the same code you saw in the last chapter.

Where it all went wrong. We counted each guess as a hit, without checking whether that cell had already been hit.

실습과제 6-1 New and improved DotCom class

```
import java.util.ArrayList;

public class DotCom {

   private ArrayList<String> locationCells;
   // private int numOfHits;
   // don't need that now

   public void setLocationCells(ArrayList<String> loc) {
      locationCells = loc;
   }

   public String checkYourself(String userInput) {

      String result = "miss";

         int index = locationCells.indexOf(userInput);

         if (index >= 0) {

            locationCells.remove(index);


            if (locationCells.isEmpty()) {
               result = "kill";
            } else {
               result = "hit";
            } // close if

         } // close outer if

      return result;
   } // close method
} // close class
```

Ignore this line for now; we talk about it at the end of the chapter.

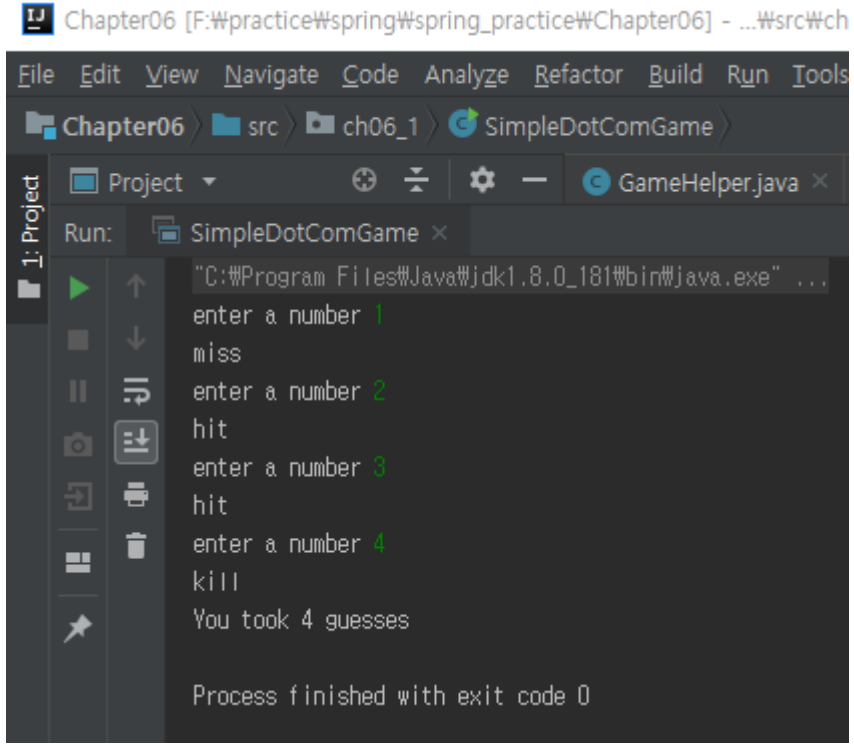
Change the int array to an ArrayList that holds Strings.

New and improved argument name.

Find out if the user guess is in the ArrayList, by asking for its index. If it's not in the list, then indexOf() returns a -1.

If index is greater than or equal to zero, the user guess is definitely in the list, so remove it.

If the list is empty, this was the killing blow!



실제 게임 “닷컴 가라앉히기” 를 만들어보자

목표: 컴퓨터가 가지고 있는 모든 닷컴명을 최소한의 추측으로 가라앉히기.
닷컴을 모두 가라앉히고 나면 성적에 따라 등급이 매겨진다.

설정: 게임이 시작되면 컴퓨터에서 닷컴 3개를 가상의 7x7 그리드 상에 배치한다.
이 작업이 끝나면 사용자에게 닷컴 위치를 추측하도록 요청한다.

게임 방법:

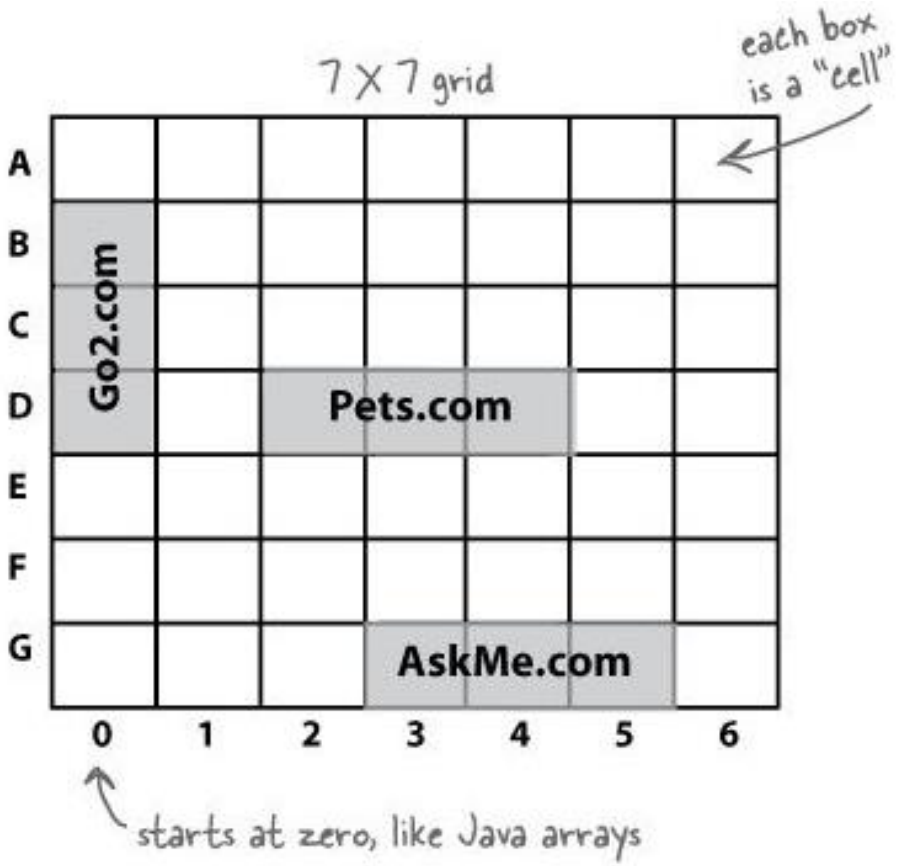
현재는 명령행에서 실행하는 버전으로 만든다. (GUI 버전은 나중에. . .)

컴퓨터에서 명령행에 위치를 추측해 보라고 프롬프트 한다.

사용자는 “A3”, “C5” 이런 식으로 명령행에 위치를 입력하면 된다.

컴퓨터에서는 맞으면 “Hit”, 틀리면 “Miss”라고 결과를 알려준다.

닷컴 3개를 모두 가라앉히면 등급이 출력된다.



게임 진행 화면

```
File Edit Window Help Sell
%java DotComBust
Enter a guess  A3
miss
Enter a guess  B2
miss
Enter a guess  C4
miss
Enter a guess  D2
hit
Enter a guess  D3
hit
Enter a guess  D4
Ouch! You sunk Pets.com   : (
kill
Enter a guess  B4
miss
Enter a guess  G3
hit
Enter a guess  G4
hit
Enter a guess  G5
Ouch! You sunk AskMe.com   : (
```

What needs to change?

변경해야 할 세 개의 클래스: **DotCom**, **DotComBust**, **GameHelper** 클래스

1. **DotCom** 클래스
   닷컴의 이름(**"Pets.com", "Go2.com"** 등)을 저장하기 위한 **name** 변수를 추가한다.

2. **DotComBust** 클래스(게임)

   이제 하나가 아닌 세 개의 닷컴을 생성한다.

   세 개의 각 닷컴에 이름을 붙인다.

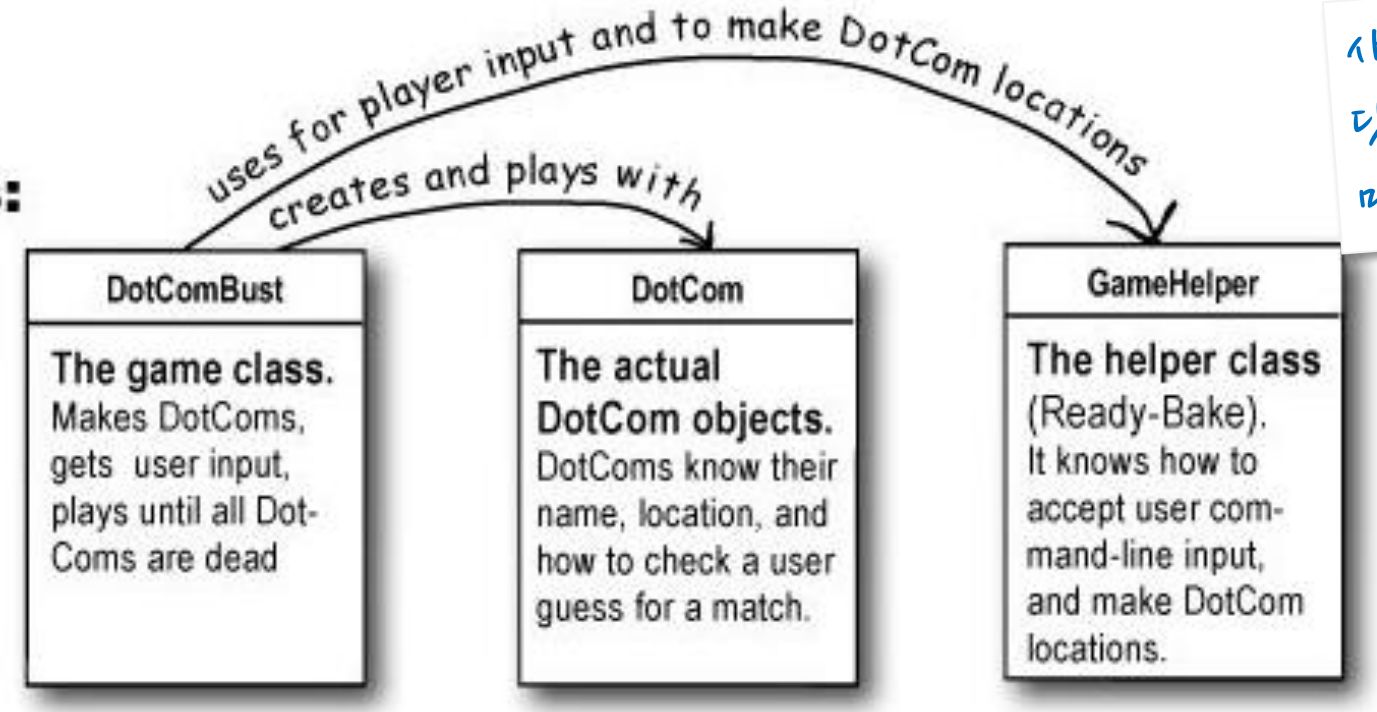
   한 행이 아닌 그리드 상에 세 개의 닷컴을 모두 배치한다.

   사용자 추측을 모든 닷컴에 대해 체크한다.

   어떠한 닷컴도 남아있지 않을 때까지 게임을 계속한다.

   main()에서 벗어난다.

3 Classes:

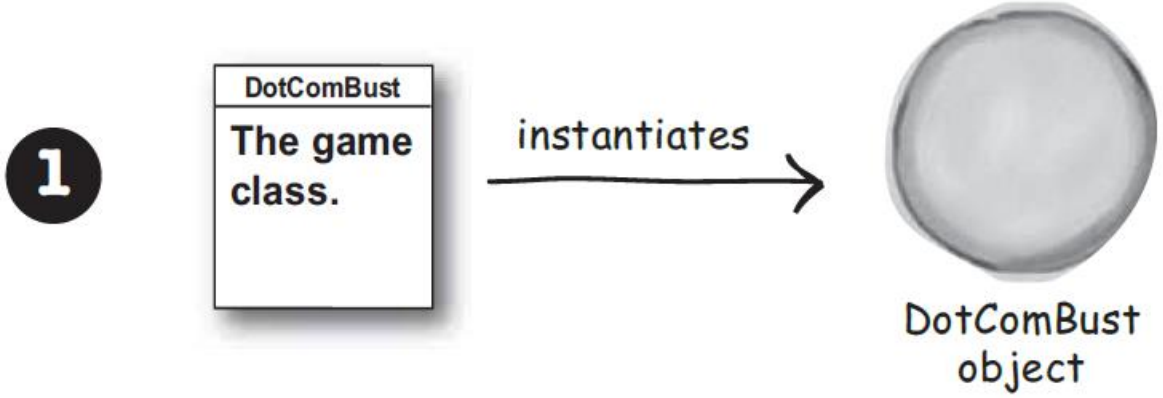


사용자 입력과
닷컴 위치를
만드는데 사용한다

5 Objects:

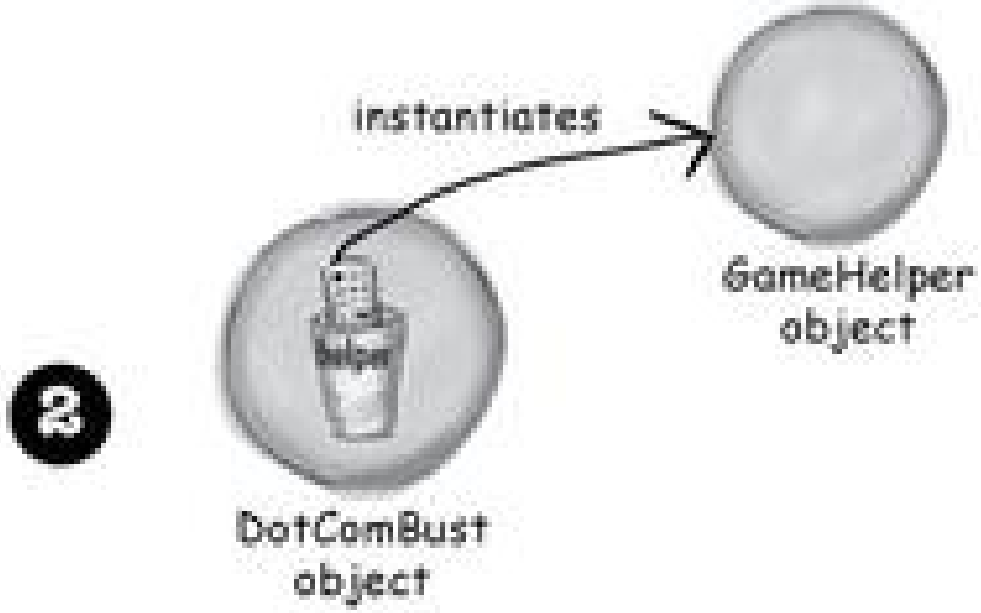
Plus 4 ArrayLists: 1 for the DotComBust and 1 for each of the 3 DotCom objects.

DotComBust 게임에서 누가 무엇을 수행하는가



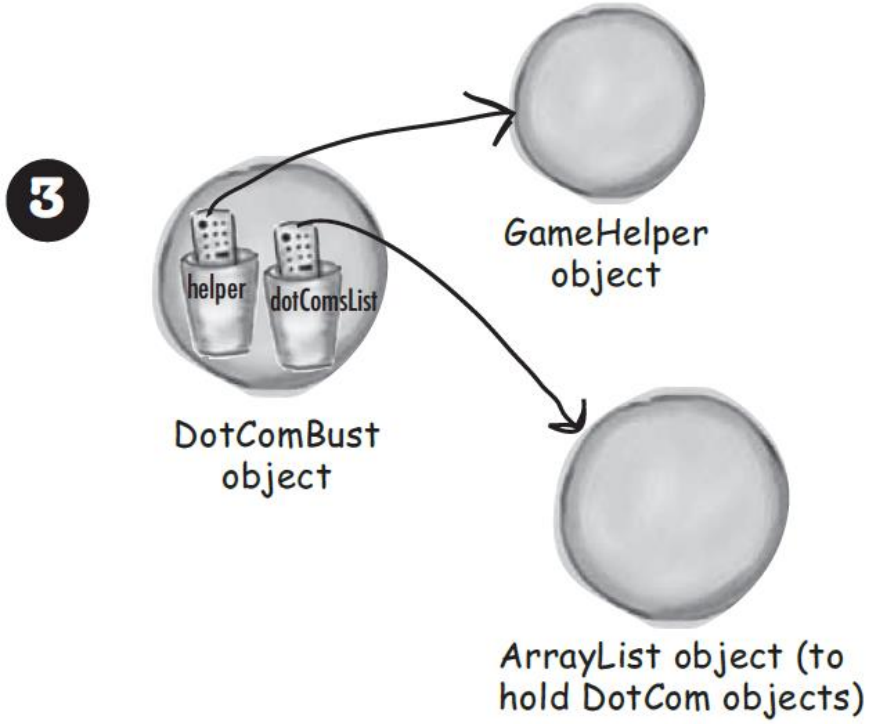
DotComBust 클래스의 **main()** 메소드

⇒ **DotComBust** 객체 생성



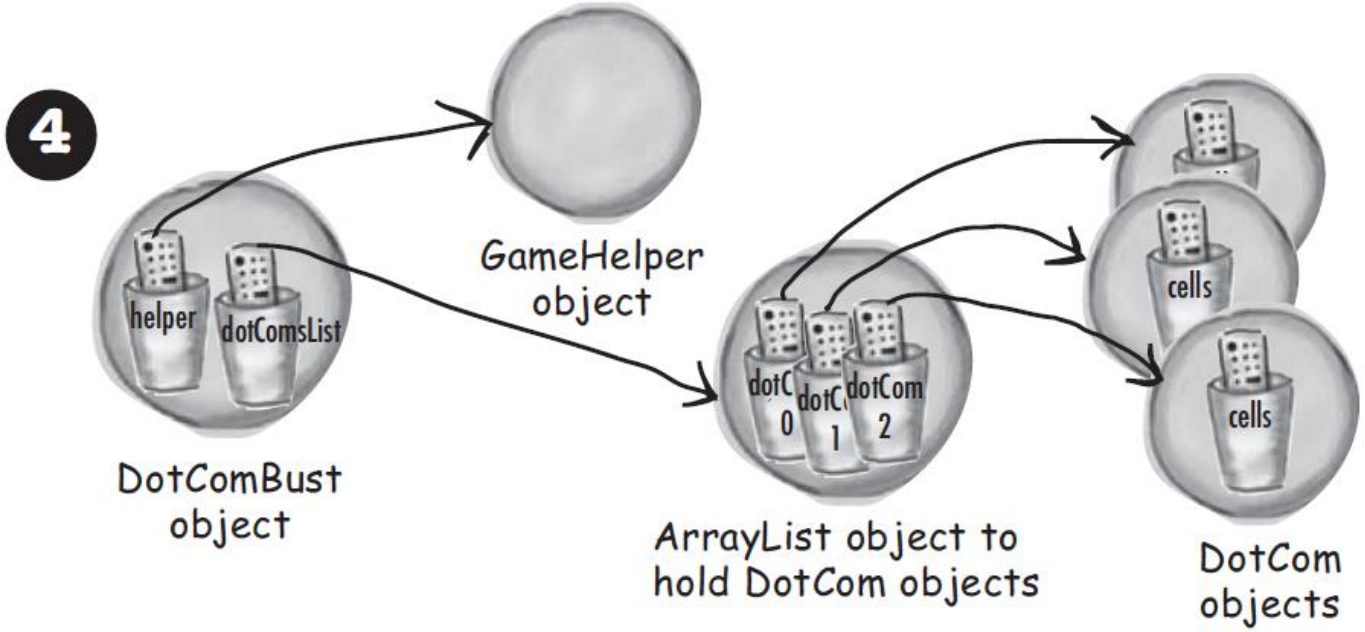
DotComBust 객체 (game)

⇒ **GameHelper** 객체 생성



DotComBust 객체

⇒ 3개의 닷컴 객체를 저장할 **ArrayList** 객체 생성



DotComBust 객체

⇒ 3개의 닷컴 객체를 생성

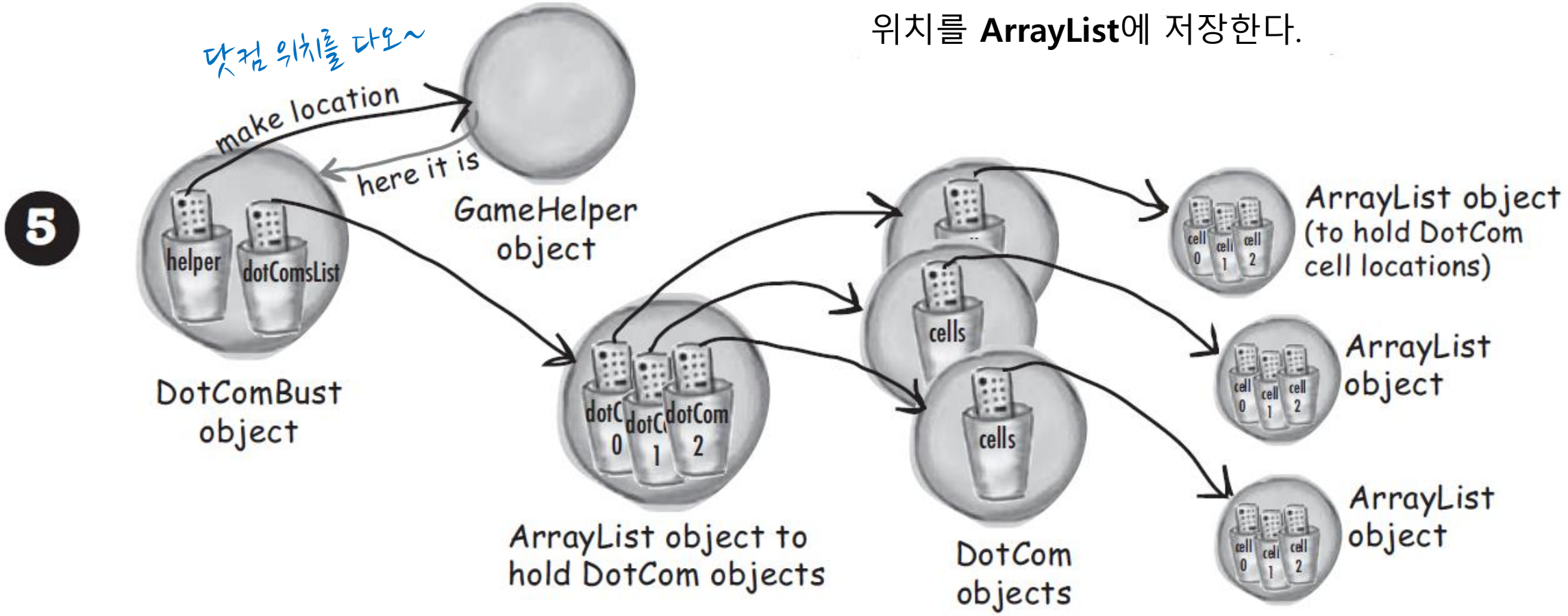
(그들을 **ArrayList**에 저장)

DotComBust 객체

⇒ **Helper** 객체에게 닷컴의 위치를 요청

DotComBust 객체

⇒ 닷컴 객체의 각각에게 위치**(like "A2", "B2")**를 제공해준다. 각 닷컴 객체는 자신의 3개의 셀 위치를 **ArrayList**에 저장한다.



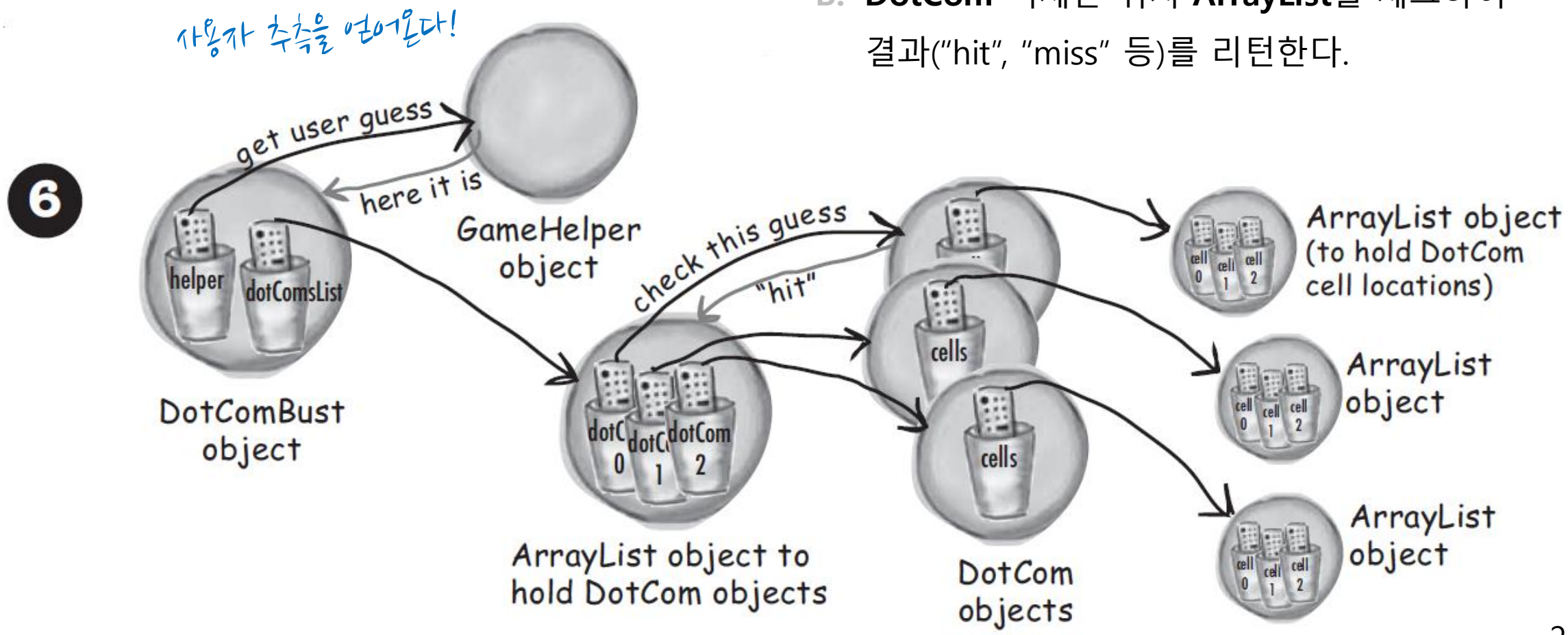
DotComBust 객체

Helper 객체에게 사용자 추측을 요청

DotComBust 객체

A. 닷컴 리스트에 대해 사용자 추측이 매치되었는지 반복해서 물어본다.

B. **DotCom** 객체는 위치 **ArrayList**를 체크하여 결과("hit", "miss" 등)를 리턴한다.



Prep code for the *real* DotComBust class

DotComBust
GameHelper helper ArrayList dotComsList int numOfGuesses
setUpGame() startPlaying() checkUserGuess() finishGame()

Variable Declarations

DECLARE and instantiate the ***GameHelper*** instance variable, named ***helper.***

DECLARE and instantiate an ***ArrayList*** to hold the list of DotComs (initially three) Call it *dotComsList*.

DECLARE an int variable to hold the number of user guesses (so that we can give the user a score at the end of the game). Name it ***numOfGuesses*** and set it to 0.

Method Declarations

DECLARE a ***setUpGame()*** method to create and initialize the DotCom objects with names and locations. Display brief instructions to the user.

DECLARE a ***startPlaying()*** method that asks the player for guesses and calls the checkUserGuess() method until all the DotCom objects are removed from play.

DECLARE a ***checkUserGuess()*** method that loops through all remaining DotCom objects and calls each DotCom object's checkYourself() method.

DECLARE a ***finishGame()*** method that prints a message about the user's performance, based on how many guesses it took to sink all of the DotCom objects.

Method Implementations

METHOD: ***void setUpGame()***

// make three DotCom objects and name them

CREATE three DotCom objects.

SET a name for each DotCom.

ADD the DotComs to the ***dotComsList*** (the ArrayList).

REPEAT with each of the DotCom objects in the ***dotComsList*** array

CALL the ***placeDotCom()*** method on the helper object, to get a randomly-selected location for this DotCom (three cells, vertically or horizontally aligned, on a 7 X 7 grid).

SET the location for each DotCom based on the result of the ***placeDotCom()*** call.

END REPEAT

END METHOD

METHOD: ***void startPlaying()***

REPEAT while any DotComs exist

GET user input by calling the helper ***getUserInput()*** method

EVALUATE the user's guess by ***checkUserGuess()*** method

END REPEAT

END METHOD

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

// find out if there's a hit (and kill) on any DotCom

INCREMENT the number of user guesses in the ***numOfGuesses*** variable

SET the local ***result*** variable (a ***String***) to "miss", assuming that the user's guess will be a miss.

REPEAT with each of the DotObjects in the ***dotComsList*** array

EVALUATE the user's guess by calling the DotCom object's ***checkYourself()*** method

SET the result variable to "hit" or "kill" if appropriate

IF the result is "kill", **REMOVE** the DotCom from the ***dotComsList***

END REPEAT

DISPLAY the ***result*** value to the user

END METHOD

METHOD: ***void finishGame()***

DISPLAY a generic "game over" message, then:

IF number of user guesses is small,

DISPLAY a congratulations message

ELSE

DISPLAY an insulting one

END IF

END METHOD

DotComBust class

```
import java.util.*;
public class DotComBust {

    private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

   private void setUpGame() {
      // first make some dot coms 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);

      System.out.println("Your goal is to sink three dot coms.");
      System.out.println("Pets.com, eToys.com, Go2.com");
      System.out.println("Try to sink them all in the fewest number of guesses");

       for (DotCom dotComToSet : dotComsList) {

          ArrayList<String> newLocation = helper.placeDotCom(3);

          dotComToSet.setLocationCells(newLocation);

      } // close for loop
   } // close setUpgame method
```

Declare and initialize the variables we'll need.

Make an ArrayList of DotCom objects (in other words, a list that will hold ONLY DotCom objects, just as DotCom[] would mean an array of DotCom objects).

Make three DotCom objects, give 'em names, and stick 'em in the ArrayList.

Print brief instructions for user.

Repeat with each DotCom in the list.

Ask the helper for a DotCom location (an ArrayList of Strings).

Call the setter method on this DotCom to give it the location you just got from the helper.

```
private void startPlaying() {

  while(!dotComsList.isEmpty()) {

     String userGuess = helper.getUserInput("Enter a guess");
     checkUserGuess(userGuess);

  } // close while
   finishGame();
} // close startPlaying method
```

As long as the DotCom list is NOT empty (the ! means NOT, it's the same as (dotComsList.isEmpty() == false).

Get user input

Call our own checkUserGuess method.

Call our own finishGame method.

```
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;
       }

    } // close for

    System.out.println(result);
 } // close method
```

increment the number of guesses the user has made

assume it's a 'miss', unless told otherwise

repeat with all DotComs in the list

ask the DotCom to check the user guess, looking for a hit (or kill)

get out of the loop early, no point in testing the others

this guy's dead, so take him out of the DotComs list then get out of the loop

Print the result for the user

print a message telling the user how he did in the game

```
    private void finishGame() {
        System.out.println("All Dot Coms are dead! Your stock is now worthless.");
        if (numOfGuesses <= 18) {
            System.out.println("It only took you " + numOfGuesses + " guesses.");
            System.out.println(" You got out before your options sank.");
        } else {
            System.out.println("Took you long enough. "+ numOfGuesses + " guesses.");
            System.out.println("Fish are dancing with your options");
        }
    } // close method


    public static void main (String[] args) {
        DotComBust game = new DotComBust();
        game.setUpGame();
        game.startPlaying();
    } // close method
}
```

create the game object

tell the game object to set up the game

tell the game object to start the main game play loop (keeps asking for user input and checking the guess)

DotCom class

```
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 userInput) {
      String result = "miss";
      int index = locationCells.indexOf(userInput);
      if (index >= 0) {
          locationCells.remove(index);

         if (locationCells.isEmpty()) {
             result = "kill";
             System.out.println("Ouch! You sunk " + name + "   : ( ");
         } else {
             result = "hit";
         }  // close if
      } // close if
       return result;

   } // close method
} // close class
```

DotCom's instance variables:
- an ArrayList of cell locations
- the DotCom's name

A setter method that updates the DotCom's location. (Random location provided by the GameHelper placeDotCom( ) method.)

Your basic setter method

The ArrayList indexOf( ) method in action! If the user guess is one of the entries in the ArrayList, indexOf( ) will return its ArrayList location. If not, indexOf( ) will return –1.

Using ArrayList's remove( ) method to delete an entry.

Using the isEmpty( ) method to see if all of the locations have been guessed

Tell the user when a DotCom has been sunk.

Return: 'miss' or 'hit' or 'kill'.

Super Powerful Boolean Expressions.

‘And’ and ‘Or’ Operators (&&, ||)

```
if (price >= 300 && price < 400) {
   camera = "X";
}
if (brand.equals("A") || brand.equals("B") ) {
   // do stuff for only brand A or brand B
}
if ((zoomType.equals("optical") &&
     (zoomDegree >= 3 && zoomDegree <= 8)) ||
     (zoomType.equals("digital") &&
     (zoomDegree >= 5 && zoomDegree <= 12))) {
   // do appropriate zoom stuff
}
```

Not equals (!= and !)

```
if (model != 2000) {
  // do non-model 2000 stuff
}
```

OR

```
if (!brand.equals("X")) {
  // do non-brand X stuff
}
```

Short Circuit Operators (&& , ||)

```
if (refVar != null &&
      refVar.isValidType() ) {
   // do 'got a valid type' stuff
}
```

refVar != null : false => refVar.isValidType() : **평가되지 않는다**

Non Short Circuit Operators (& , |)

When used in boolean expressions, the & and | operators act like their && and || counterparts, except that they force the JVM to **always check both sides** of the expression. Typically, & and | are used in another context, for manipulating bits.

GameHelper.java (1/4)

```
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 getUserInput(String prompt) {
     String inputLine = null;
     System.out.print(prompt + "  ");
     try {
       BufferedReader is = new BufferedReader(
        new InputStreamReader(System.in));
       inputLine = is.readLine();
       if (inputLine.length() == 0 )  return null;
     } catch (IOException e) {
       System.out.println("IOException: " + e);
     }
     return inputLine.toLowerCase();
  }
```

GameHelper.java (2/4)

```
public ArrayList<String> placeDotCom(int comSize) {
   ArrayList<String> alphaCells = new ArrayList<String>();
                                                    // holds 'f6' type coords
   String temp = null;                              // temporary String for concat
   int [] coords = new int[comSize];                // current candidate coords
   int attempts = 0;                                // current attempts counter
   boolean success = false;                         // flag = found a good location ?
   int location = 0;                                // current starting location

   comCount++;                                      // nth dot com to place
   int incr = 1;                                    // set horizontal increment
   if ((comCount % 2) == 1) {                       // if odd dot com (place vertically)
     incr = gridLength;                             // set vertical increment
   }

   while ( !success & attempts++ < 200 ) {           // main search loop  (32)
      location = (int) (Math.random() * gridSize);   // get random starting point
       //System.out.print(" try " + location);
      int x = 0;                                     // nth position in dotcom to place
       success = true;                               // assume success
       while (success && x < comSize) {              // look for adjacent unused spots
         if (grid[location] == 0) {                  // if not already used
```

GameHelper.java (3/4)

```
          coords[x++] = location;                          // save location
          location += incr;                                // try 'next' adjacent
          if (location >= gridSize){                       // out of bounds - 'bottom'
            success = false;                               // failure
          }
          if (x>0 && (location % gridLength == 0)) {  // out of bounds - right edge
            success = false;                               // failure
          }
        } else {                                           // found already used location
           // System.out.print(" used " + location);
           success = false;                                // failure
        }
      }
}                                                          // end while

int x = 0;                                                 // turn location into alpha coords
int row = 0;
int column = 0;
// System.out.println("\n");
while (x < comSize) {
  grid[coords[x]] = 1;                                     // mark master grid pts. as 'used'
  row = (int) (coords[x] / gridLength);                    // get row value
  column = coords[x] % gridLength;                         // get numeric column value
  temp = String.valueOf(alphabet.charAt(column));          // convert to alpha
```

GameHelper.java (4/4)

```
        alphaCells.add(temp.concat(Integer.toString(row)));
        x++;
        // System.out.print("  coord "+x+" = " + alphaCells.get(x-1));
      }

      // System.out.println("\n");

      return alphaCells;
    }
  }
```

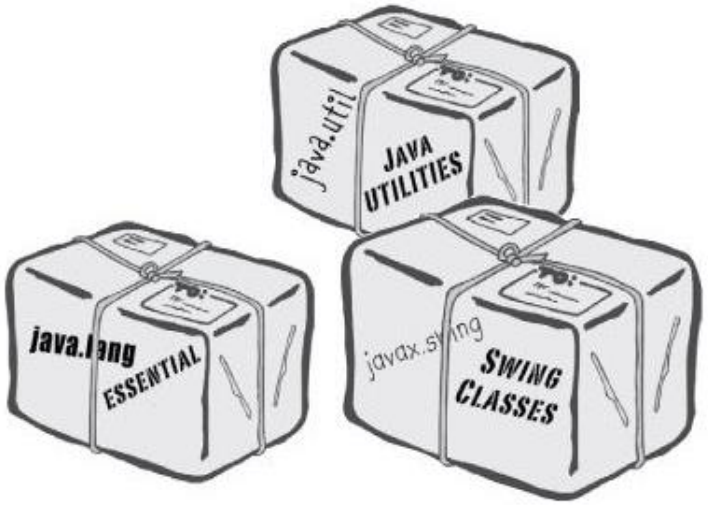
This is the statement that tells you exactly where the DotCom is located.

실습과제 6-2 DotComBust game

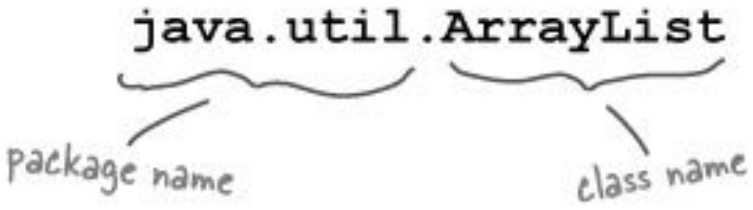
```
File Edit Window Help Sell
%java DotComBust
Enter a guess  A3
miss
Enter a guess  B2
miss
Enter a guess  C4
miss
Enter a guess  D2
hit
Enter a guess  D3
hit
Enter a guess  D4
Ouch! You sunk Pets.com   : (
kill
Enter a guess  B4
miss
Enter a guess  G3
hit
Enter a guess  G4
hit
Enter a guess  G5
Ouch! You sunk AskMe.com   : (
```

라이브러리 이용 (Java API)

자바 API에서 클래스는 패키지로 묶여진다.



API의 클래스를 사용하기 위해서는 그 클래스가 어느 패키지에 속하는지를 알아야 한다.
코드에서는 클래스의 풀네임을 알아야만 한다:



How to play with the API

http://docs.oracle.com/javase/8/docs/api/

Java™ Platform Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer
AbstractPreferences
AbstractProcessor
AbstractQueue
AbstractQueuedLongSynchronizer

http://docs.oracle.com/javase/8/docs/api/javax/swing/AbstractAction.html

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.

실습과제 6-3 Code Magnets

```
File Edit Window Help Dance
% java ArrayListMagnet
zero  one  two  three
zero  one  three  four
zero  one  three  four  4.2
zero  one  three  four  4.2
```

```
a.remove(2);
printAL(a);
printAL(a);
a.add(0,"zero");
a.add(1,"one");
public static void printAL(ArrayList<String> al) {
if (a.contains("two")) {
    a.add("2.2");
}
a.add(2,"two");
public static void main (String[] args) {
System.out.print(element + "  ");
}
System.out.println(" ");
if (a.contains("three")) {
    a.add("four");
}
public class ArrayListMagnet {
if (a.indexOf("four") != 4) {
    a.add(4, "4.2");
}
}
}
}
import java.util.*;
printAL(a);
ArrayList<String> a = new ArrayList<String>();
for (String element : al) {
a.add(3,"three");
printAL(a);
```

실습과제 6-4 피타고라스 정리

피타고라스의 정리는 직각 삼각형에서 직각을 낀 두 변의 길이를 a, b라고 하고, 빗변의 길이를 c라고 하면 $a^2 + b^2 = c^2$의 수식이 성립한다는 것이다.

각 변의 길이가 100보다 작은 삼각형 중에서 피타고라스의 정리가 성립하는 직각 삼각형은 몇 개나 있을까?

3중 반복문을 이용하여 피타고라스의 정리를 만족하는 3개의 정수를 찾도록 한다.