Chapter 6

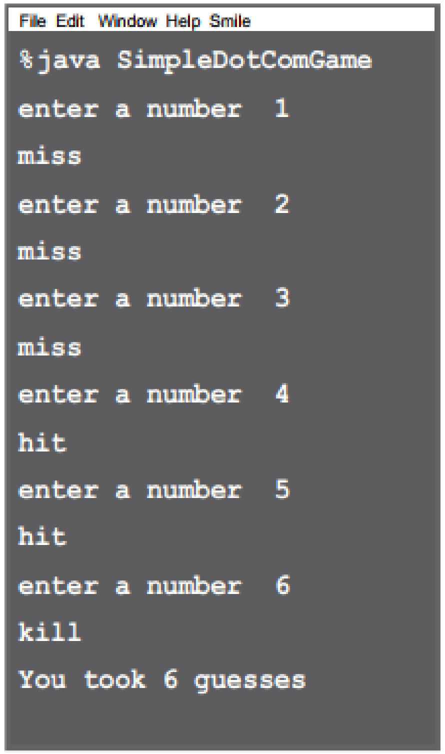
Using the Java Library (Java API)

- Recall the program we wrote in the last chapter.

How it's supposed to look

Here's what happens when we run it and enter the numbers 1,2,3,4,5,6. Lookin' good.

A complete game interaction
(your mileage may vary)

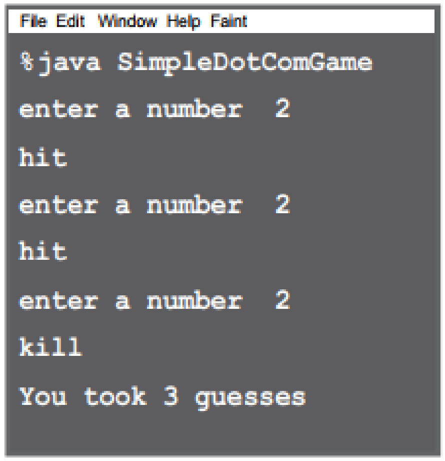


```
%java SimpleDotComGame
enter a number  1
miss
enter a number  2
miss
enter a number  3
miss
enter a number  4
hit
enter a number  5
hit
enter a number  6
kill
You took 6 guesses
```

How the bug looks

Here's what happens when we enter 2,2,2.

A different game interaction
(yikes)



```
%java SimpleDotComGame
enter a number  2
hit
enter a number  2
hit
enter a number  2
kill
You took 3 guesses
```

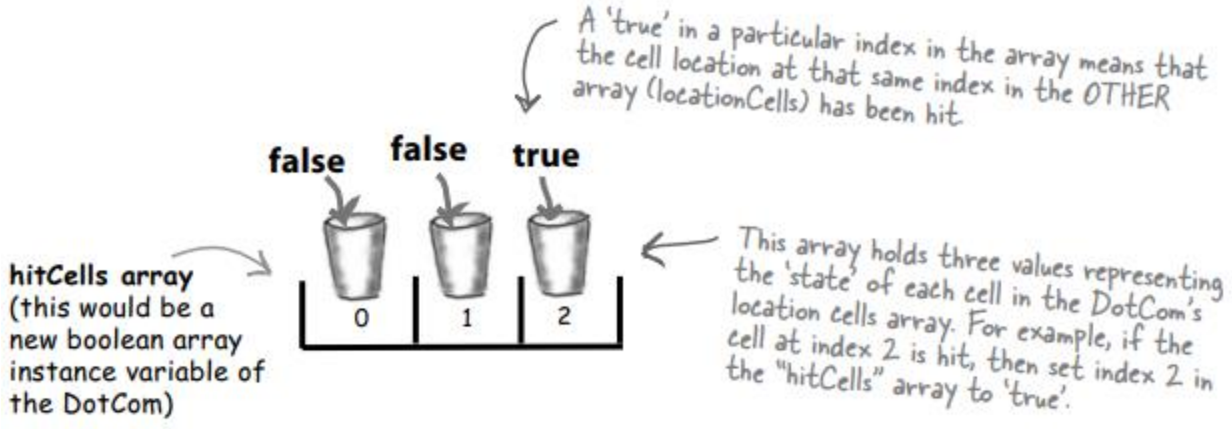
In the current version, once you get a hit, you can simply repeat that hit two more times for the kill!

- Where is the bug?

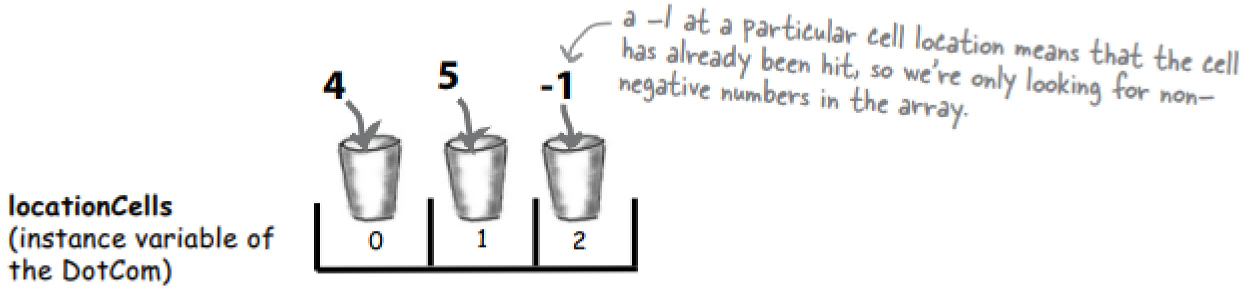
```
public String checkYourself(String stringGuess) {
    int guess = Integer.parseInt(stringGuess);
    String result = "miss";
    for (int cell: locationCells)
    {
        if (guess == cell) {
            result = "hit";
            numOfHits++;
            break;
        }
    }
    if (numOfHits == locationCells.length)
    {
        result = "kill";
    }
    System.out.println(result);
    return result;
}
```

We counted a hit every time even if that location had already been hit!

- How do we fix it?
- We need a way to know whether a cell has already been hit
- Option 1
  - Make a second array, and each time the user makes a hit, we set the value to true

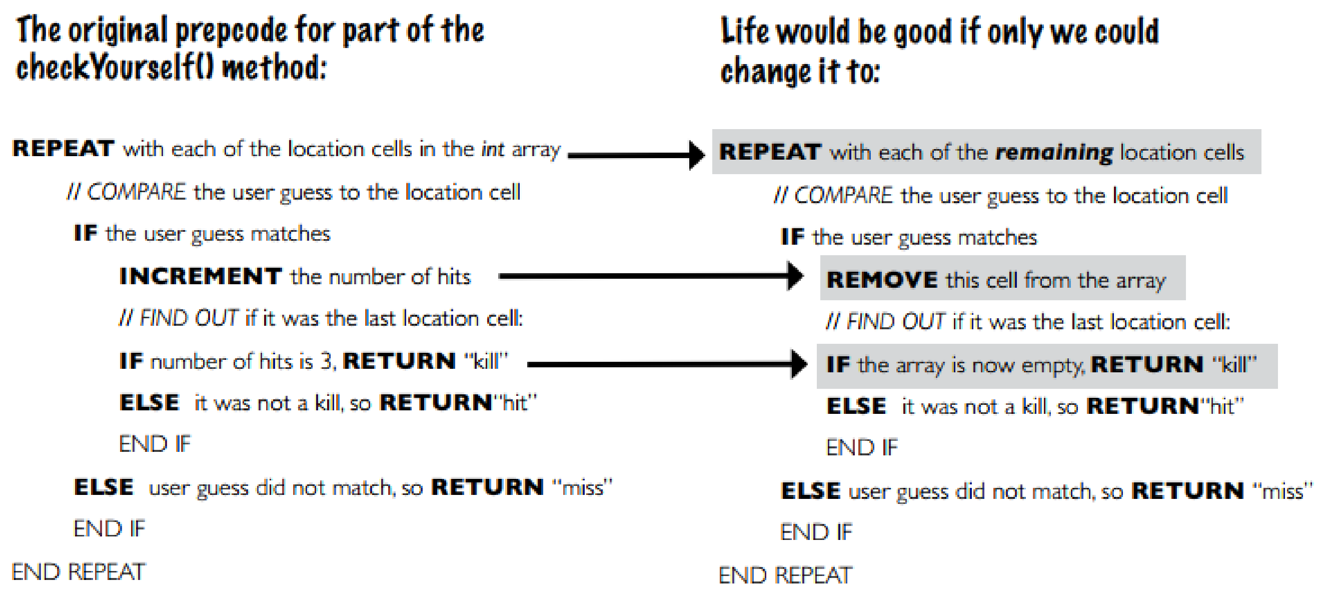

A 'true' in a particular index in the array means that the cell location at that same index in the OTHER array (locationCells) has been hit.
false
false
true
hitCells array (this would be a new boolean array instance variable of the DotCom)
0
1
2
This array holds three values representing the 'state' of each cell in the DotCom's location cells array. For example, if the cell at index 2 is hit, then set index 2 in the "hitCells" array to 'true'.

- Can we find anything better?
- Option 2
- Keep the one original array, but change the value of any hit cells to -1


a -1 at a particular cell location means that the cell has already been hit, so we're only looking for non-negative numbers in the array.
4
5
-1
0
1
2
locationCells (instance variable of the DotCom)

- Option 2 is a little less clunky than option one, but it's not very efficient. You'd still have to loop through all three slots (index positions) in the array

- Can we find anything better?
- Option 3
  - We delete each cell location as it gets hit, and then modify the array to be smaller.
  - However, arrays can’t change their size, so we have to make a new array and copy the remaining cells from the old array into the new smaller array



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
```

- What you want to have:
- An array that could shrink when you remove something.
- An array or something that we can ask it if it contains what you’re looking for without looping through it
- Is that possible?
- At anytime you can write a tool class which is able to perform this task, for example

```
MyArrayTool().removeElement(arr, idx);
```

- Exercise
- Try to see if you can complete following tasks

Mission	Code to do that
1. I want to have an array of ints. Initially it is an empty array	Int[] arr = new int[0];
2. I want to put an int into this array	
3. I want to put another int into this array	
4. I want to remove the last element into this array	
5. I want to know the number of elements in this array	

- In fact, you don't have to reinvent the wheel if you know how to find what you need in the Java library, known as the **Java API**
- The core Java library is a giant pile of classes just waiting for you to use like building blocks, to assemble your own program out of largely pre-built code.

- There really is such a thing in Java, called **ArrayList**.
- The ArrayList is a class in the core Java library (the API)
- p.s. The ArrayList belongs to a very important framework in Java called ***collections***. You'll learn this in the latter chapter.

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

① Make one

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

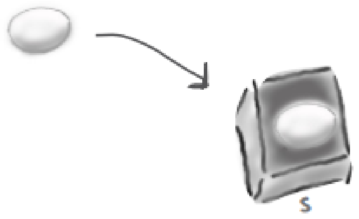
```
ArrayList<Egg> myList = new ArrayList<Egg>();
```

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);
```

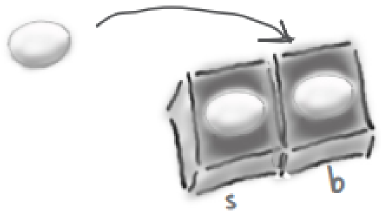


Now the ArrayList grows a "box" to hold the Egg object.

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

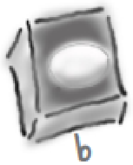
⑦ Find out if it's empty

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

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

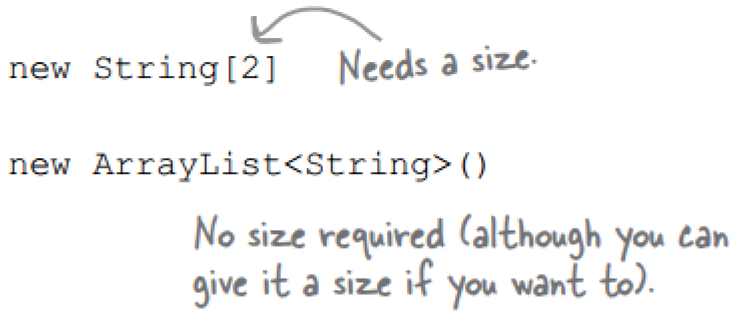
⑧ Remove something from it

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



Hey look — it shrank!

- Comparing ArrayList to a regular array
- **Size**
  - Regular array is a ***fixed sized*** array
  - ArrayList is like a ***dynamic*** array i.e. we don't need to declare its size, it grows as we add elements to it and it shrinks as you remove elements from it, during the runtime of the program.



```
new String[2]
```

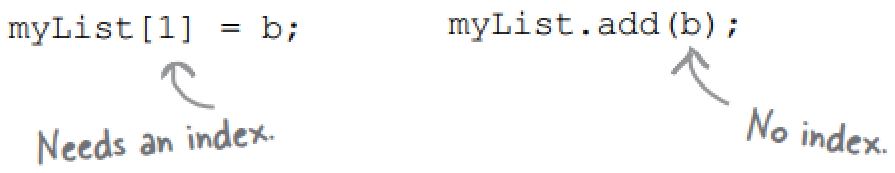
Needs a size.

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

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

- **Variables**
  - Regular array can contain both primitives and objects
  - ArrayList can contain only ***objects***

- Comparing ArrayList to a regular array
- **Add/remove variables**
  - Array use indices to store elements – objects are not actually removed
  - ArrayList use **add()** to insert elements.



```
myList[1] = b;        myList.add(b);
```

- **Performance**
  - Array is fast
  - ArrayList is less-efficient but more generic
- Even though an array is an object, it lives in its own special world and you can't invoke any methods on it.

```
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.

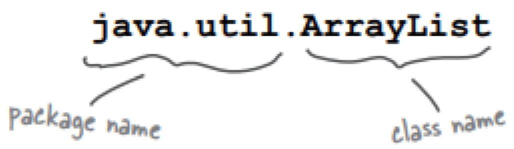
Find out if the user guess is in the ArrayList, by asking for its index. If it's not in the list, then indexOf() returns a -1.

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!

- About Java APIs
- In the Java API, classes are grouped into **packages**.
- To use a class in the API, you have to know which package the class is in.
- Every class in the Java library belongs to a package. The package has a name, like **javax.swing** (a package that holds some of the Swing GUI classes).
- ArrayList is in the package called **java.util**, which holds a pile of utility classes.
- So far, you've already been using classes from a package such as System (System.out.println), String, and Math (Math.random()), all belong to the **java.lang** package.

- You have to know the ***full*** name of the class you want to use in your code, and that means package name + class name.
- The full name of ArrayList is actually:



- You have to tell Java which ArrayList you want to use. You have two options:
- Put an **import** statement at the top of your source code file

```
import java.util.ArrayList;
```

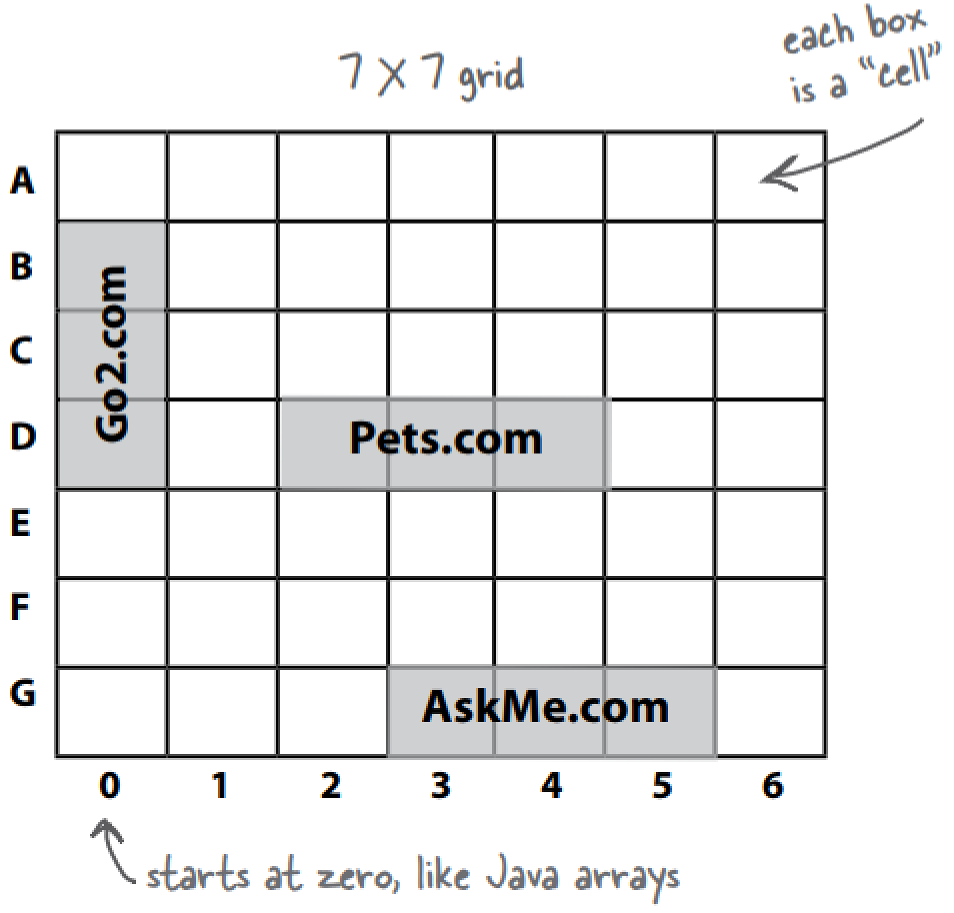
Or

- Type the full name each time you use it.

```
java.util.ArrayList<Dog> list = new java.util.ArrayList<Dog>();
```

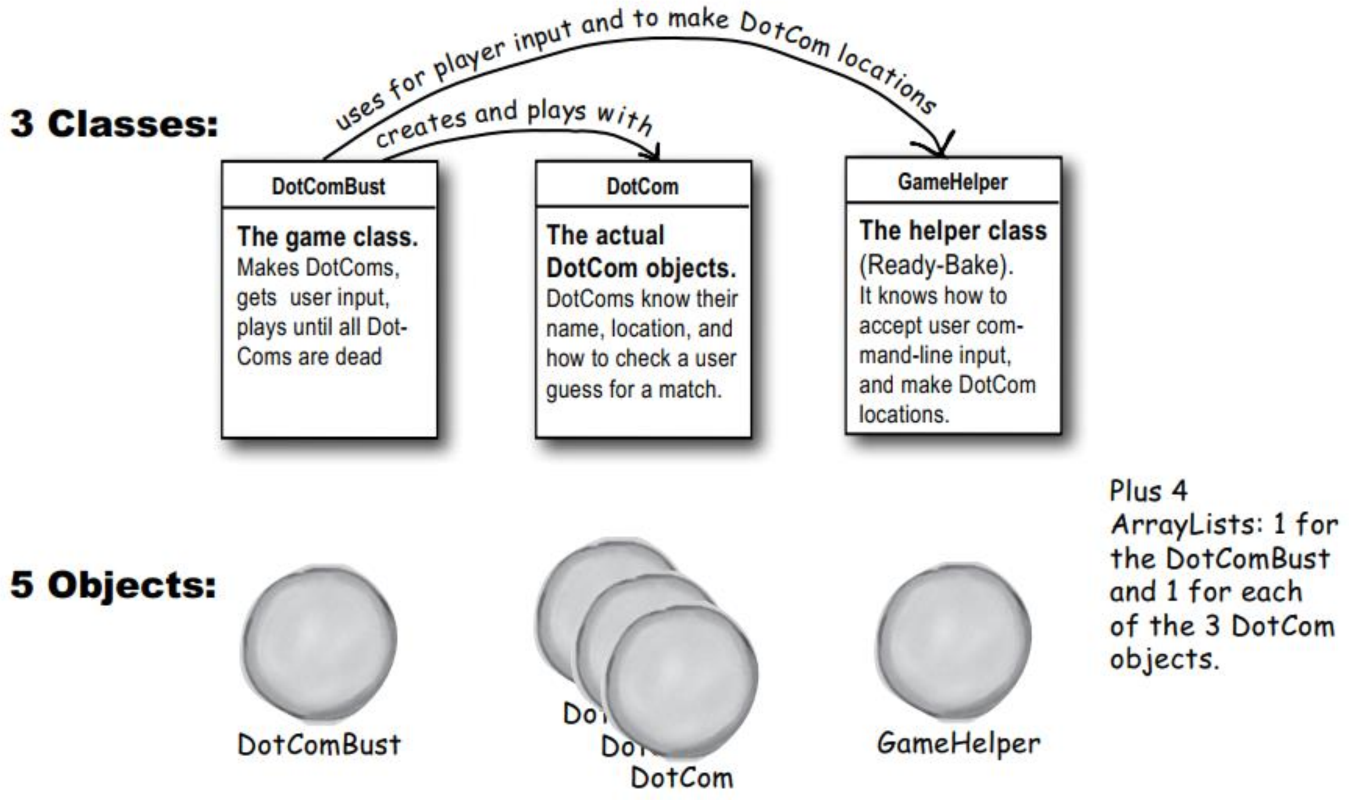
- Why does there have to be a full name?
- Packages are important for three main reasons.
- First, they help the overall organization of a project or library. Rather than just having one horrendously large pile of classes, they're all grouped into packages for specific kinds of functionality (like GUI, or data structures, or database stuff, etc.)
- Second, packages give you a *namescoping*, to help prevent collisions if you and other programmers in your company all decide to make a class with the same name. If you have a class named Set and someone else (including the Java API) has a class named Set, you need some way to tell the JVM which Set class you're trying to use.
- Third, packages provide a level of security, because you can restrict the code you write so that only other classes in the same package can access it.

- We’ve been working on the `simple’ version, but now let’s build the real one. Instead of a single row, we’ll use a grid. And instead of one DotCom, we’ll use three.

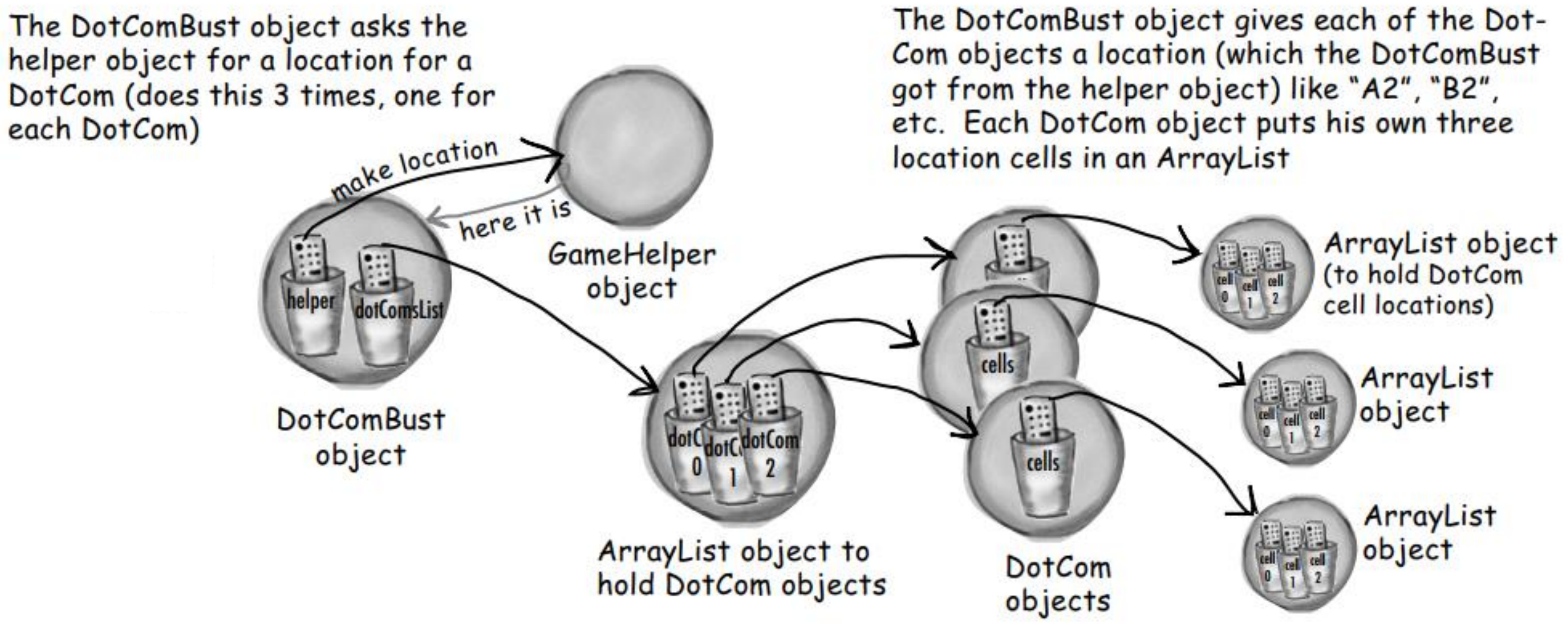


- We have three classes that need to change: the DotCom class (which is now called DotCom instead of SimpleDotCom), the game class (DotComBust) and the game helper class (which we won't worry about now).
- DotCom class
  - Add a name variable to hold the name of the DotCom ("Pets.com", "Go2.com", etc.) so each DotCom can print its name when it's killed (see the output screen on the opposite page).
- DotComBust class
  - Create three DotComs instead of one.
  - Give each of the three DotComs a name.
  - Call a setter method on each DotCom instance, so that the DotCom can assign the name to its name instance variable.
  - Put the DotComs on a grid rather than just a single row, and do it for all three DotComs.

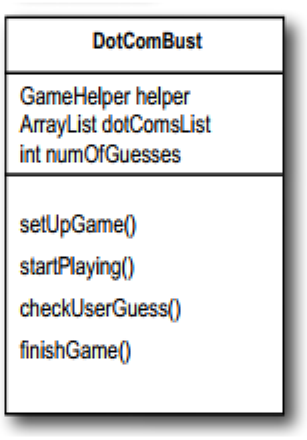
- Check each user guess with all three DotComs, instead of just one.
- Keep playing the game (i.e. accepting user guesses and checking them with the remaining DotComs) until there are no more live DotComs.


3 Classes:
uses for player input and to make DotCom locations
creates and plays with
DotComBust
The game class. Makes DotComs, gets user input, plays until all Dot-Coms are dead
DotCom
The actual DotCom objects. DotComs know their name, location, and how to check a user guess for a match.
GameHelper
The helper class (Ready-Bake). It knows how to accept user com-mand-line input, and make DotCom locations.
5 Objects:
DotComBust
DotCom
GameHelper
Plus 4 ArrayLists: 1 for the DotComBust and 1 for each of the 3 DotCom objects.

- DS in Game


The DotComBust object asks the helper object for a location for a DotCom (does this 3 times, one for each DotCom)
make location
here it is
GameHelper object
helper
dotComsList
DotComBust object
The DotComBust object gives each of the Dot-Com objects a location (which the DotComBust got from the helper object) like "A2", "B2", etc. Each DotCom object puts his own three location cells in an ArrayList
dotC 0
dotC 1
dotCom 2
ArrayList object to hold DotCom objects
cells
cells
DotCom objects
cell 0
cell 1
cell 2
ArrayList object (to hold DotCom cell locations)
ArrayList object
ArrayList object

• Pseudo code



DotComBust

GameHelper helper
ArrayList dotComsList
int numOfGuesses

setUpGame()
startPlaying()
checkUserGuess()
finishGame()

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.

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: 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
```

- Pseudo code

```
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
```

```
public class DotComBust {
    private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

    private void setUpGame() {
        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 dotComSet : dotComsList) {
            ArrayList<String> newLocation = helper.placeDotCom(3);
            dotComSet.setLocationCells(newLocation);
        }
    }

    private void startPlaying() {
        while (!dotComsList.isEmpty()) {
            String userGuess = helper.getUserInput("Enter a guess");
            checkUserGuess(userGuess);
        }
        finishGame();
    }
```

Can you match the annotations at the bottom with the code?

ask the helper for a DotCom location

get user input

repeat with each DotCom in the list

declare and initialize the variables we'll need

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

call our own checkUserGuess method

print brief instructions for user

as long as the DotCom list is NOT empty

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

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;
        }
    }
    System.out.println(result);
}

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.");
    }
}
```

中原大學
Chung Yuan Christian University

repeat with all DotComs in the list

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

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

Print the result for the user

tell the game object to set up the game

increment the number of guesses the user has made

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

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

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

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

create the game object

```
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'.

- Recall: boolean expressions
- The boolean expressions in Java are very similar to that in C
- and: &&
- or: ||
- not equal: !=
- not: !
- short circuit operators: && , ||
    - JVM sees that the left side of a && expression is false, it won't check the expression at the right side.
- non short circuit: & , |
    - JVM always check both sides of &&/|| expressions

```
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();
  }

 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
```

Note: For extra credit, you might try 'un-commenting' the System.out.print(ln)'s in the placeDotCom( ) method, just to watch it work! These print statements will let you "cheat" by giving you the location of the DotComs, but it will help you test it.

```
          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

      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.

- Summary
- ArrayList is a class in the Java API
- Basic operations of an ArrayList: add(), remove(), indexOf(), isEmpty(), size()
- An ArrayList only holds objects of same type. You can't put primitives in an ArrayList
- Every class in Java belongs to a package; If you create a Java file without any package name and compile it, the resulting class file will have a package name which is also known as default package
- You can use the import statement to tell the Java compiler which class you're going to use or type the full name every place you use the class in your code.

- Does import make my class bigger? Does it actually compile the imported class or package into my code?
- An import in Java is not the same as an include in C. It is simply the way you give Java the full name of a class.
- Since System, String, Math are all come from java.lang, why I don't have to import them before using them?
- Because all classes in the java.lang package are imported by default.
- Can I put my own class into packages?
- Yes, you will want to put your classes into packages (in real applications). We'll get into that in detail in the latter chapter.

- There are two things you may want to know
- What classes are in the library?
- Once you find a class, how do you know what it can do?
- I suggest you to use the HTML API docs
- Java 8
- https://docs.oracle.com/javase/8/docs/api/
- Java 10
- https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html