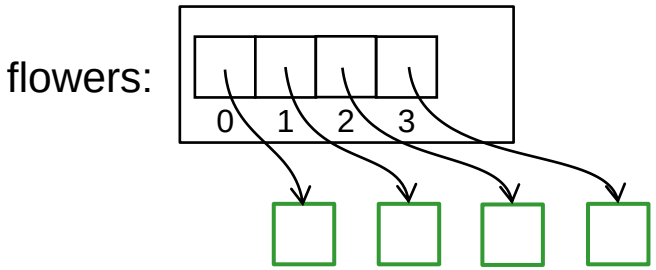
Saving and loading lists of objects

- Save a garden of flowers to a file to load later:
  - Get file name and open PrintStream to file
  - step through list of flowers, printing info to file



- Load a garden of flowers from a file
  - Make an empty list
  - Get file name and read all the lines.
  - for each line,
    - extract the values
    - create new Flower
    - add to the list.

```
104  268  40  bud
287 132  70  bloom
524 245  20  picked
274  83  50  bud
```

Saving and Loading objects

- Turn Object into text in a file that can be read in order to reconstruct the Object.

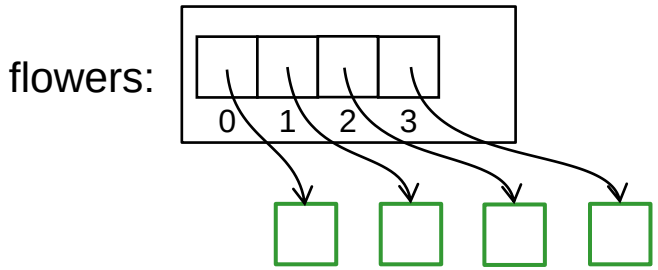
eg, for the Flower class:

toString()
- a standard method for all Objects.
- println knows about it.

```
/** Returns a String representation of the Flower, suitable for saving to a file */
public String toString(){
    return ""+this.baseX+" "+this.baseY+" "+this.height+" "+ this.stage;
}

/** Constructor #2 : Makes a new Flower with the specified values.  */
public Flower(double x, double y, double ht, String st){
    this.baseX = x;
    this.baseY = y;
    this.height = ht;
    this.stage = st;
    this.draw();
}
```

Saving Garden to a file

- Save a garden of flowers to a file to load later:
  - Get file name and open file
  - step through flowers, printing to file



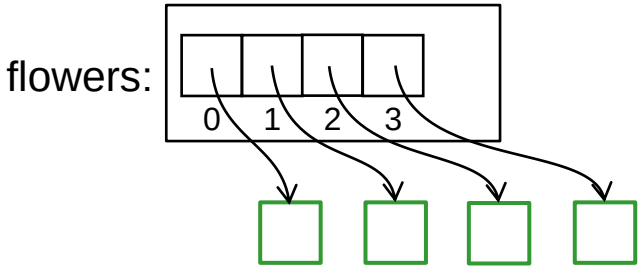

```
public void doSave(){
    try{
        PrintStream out = new PrintStream(UIFileChooser.save("File for Garden"));
        for (Flower flower : this.flowers) {
            out.println(flower.toString());        or just        out.println(flower);
        }
        out.close();
    }
    catch(IOException e) { UI.println("File saving failed: "+e); }
}
```

toString() method called automatically

Loading Garden from a file

- Get file name and open file
- Step through file, reading

```
104 268  40  bud
287 132  70  bloom
524 245  20  picked
274  83  50  bud
          :
```




```
public void load(){
    List<String> lines = Files.readAllLines(Path.of(UIFileChooser.open("Garden File")));
    this.flowers.clear();              // or  this.flowers = new ArrayList<Flower>();
    for (String line : lines){
        Scanner sc = new Scanner(line);
        double x = sc.nextDouble();
        double y = sc.nextDouble();
        double height = sc.nextDouble();
        String stage = sc.next();
        this.flowers.add(new Flower(x, y, height, stage) );
    }
}
```

Flower Class

```
import ecs100.*;
import java.awt.Color;

public class Flower{
    // fields
    private double baseX;
    private double baseY;
    private double height;
    private String stage;

    /** constructor for planting; given position */
    public Flower(double x, double y){
            this.baseX = x;
            this.baseY = y;
            this.stage = "Bud";
            this.height = 20;
    }
```

```
    /** additional constructor; for reconstructing flower */
    public Flower(double x, double y, double h, String s){
        this.baseX = x;
        this.baseY = y;
        this.stage = s;
        this.height = h;}


    /** returns text description of the Flower*/
    public String toString(){
        return ( this.baseX + " " + this.baseY + " " +
                    this.height + " " + this.stage );


     …
```

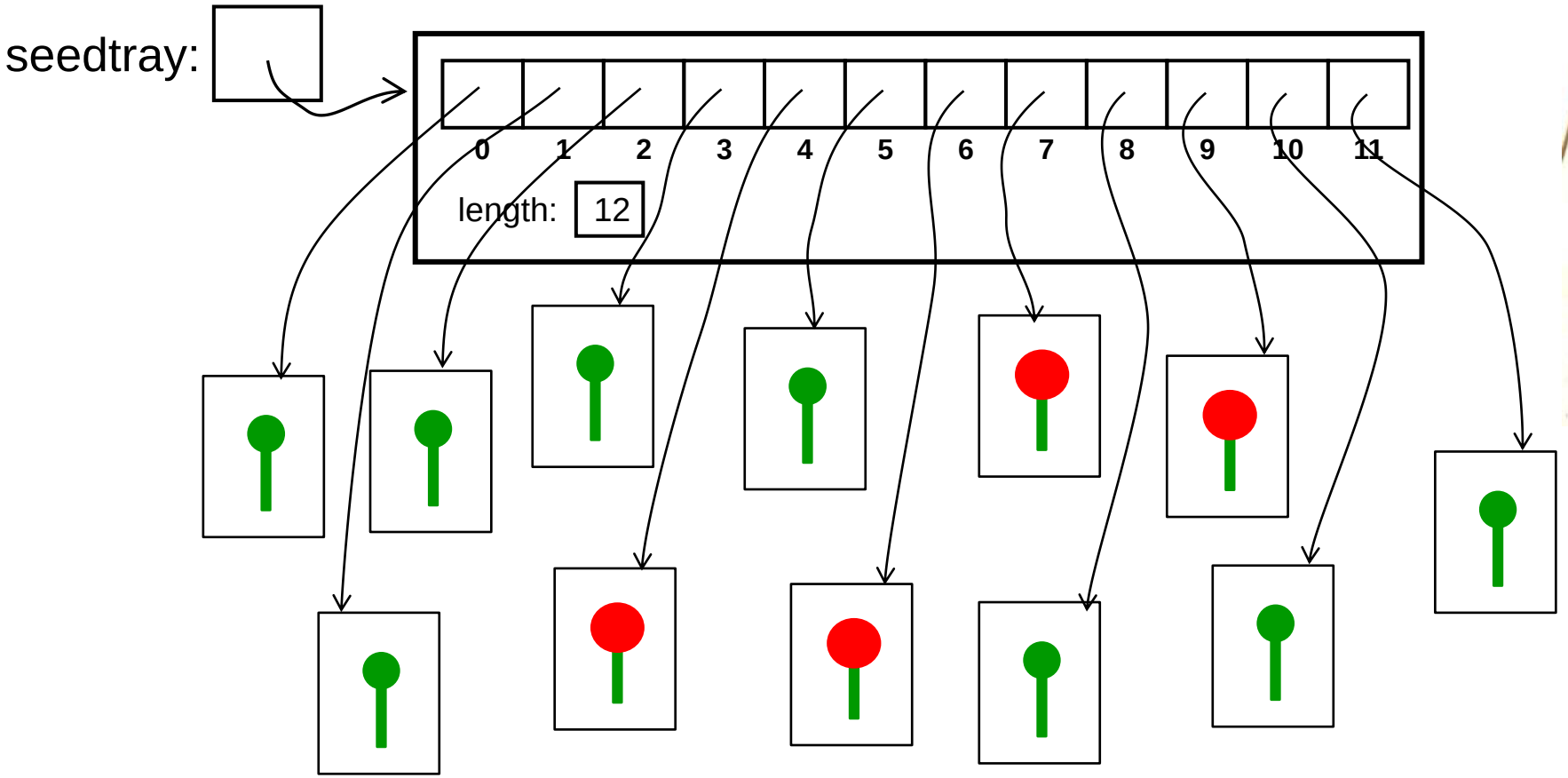
Arrays vs ArrayLists

- Some lists have a fixed number of places:

- The places may be empty

- Arrays may be sometimes more convenient than using ArrayLists

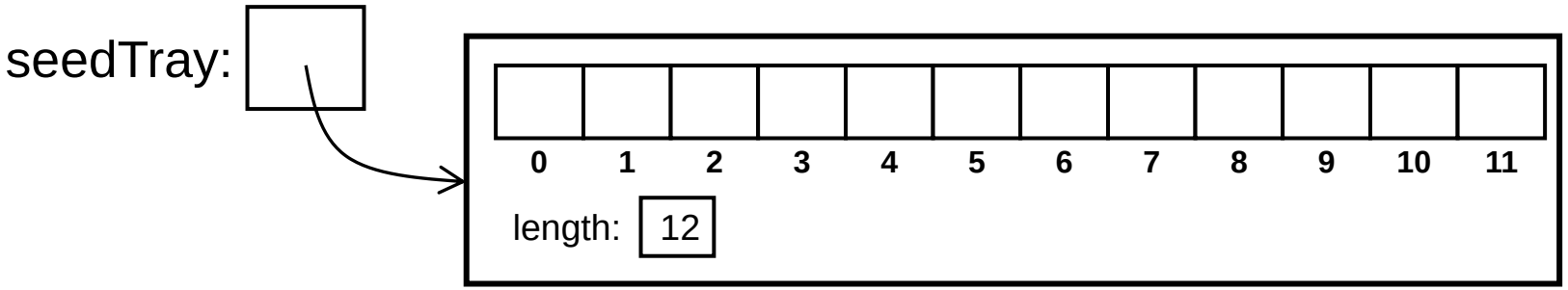
SeedTray Program: just 12 flowers



from http://bifurcatedcarrots.eu/wp-content/uploads/2010/04/tps_seedlings.jpg

Arrays

- An array is an object with a fixed number of places
  - Length determined when array is created
  - All elements are of the same type
  - Special syntax, no methods



  - Each element specified by its *index* (an int expression)

    seedtray[4] ← name of the element of shapes with index 4

    seedtray[n-3]

    - Counting from 0, just like ArrayLists!

  - Array knows its length: seedtray.length

Confusion:
names.size() ← ArrayList
name.length() ← String
tray.length ← Array

Declaring and Creating Arrays

- Declare a variable to hold an array object by putting [ ] after the type of the elements:

```
Flower[ ]  seedtray;
String[ ] keywords;
private double[ ] marks;
```

Creates a place that can hold an array
Doesn't create the array itself

- Create an array object with **new** and the length of the array:

```
new Flower[12];
new String[50];
new double[200];
```

NO ROUND BRACKETS !!!

Creates an array object, but nothing in it

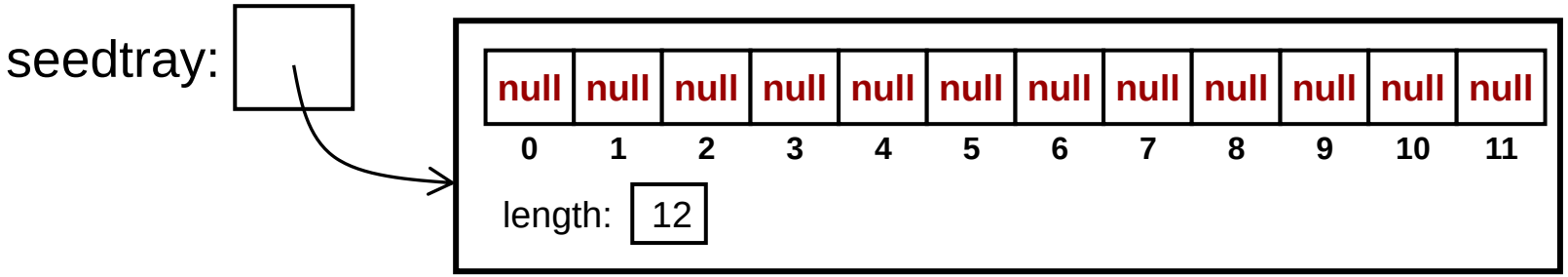
Can have an array of double or int (unlike ArrayLists)

- As usual, can combine declaration and initialisation:
  - String [ ] keywords = **new** String [50];

What does the new array contain?

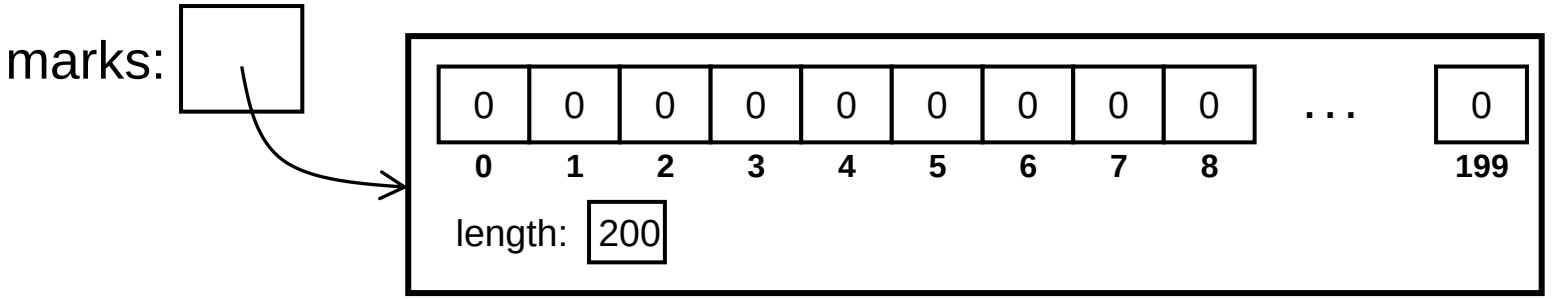
Initial values in a new array

```
Flower[ ]  seedtray =  new Flower[12];
```



Arrays of objects initialised with **null** (the "no object here" value)

```
double[ ]  marks =  new double[200];
```

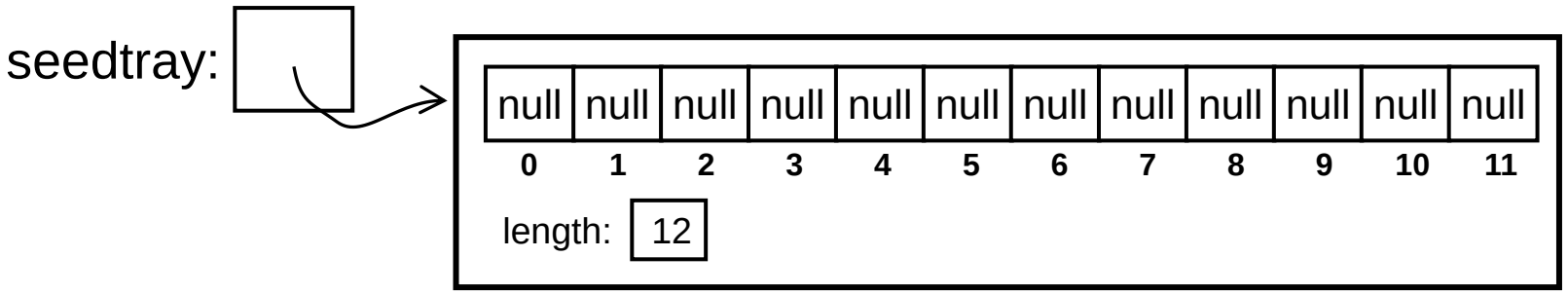


Arrays of numbers initialised to 0.

SeedTray Program

```
public class SeedTray {

    private  Flower[ ] seedtray = new  Flower[12];
```

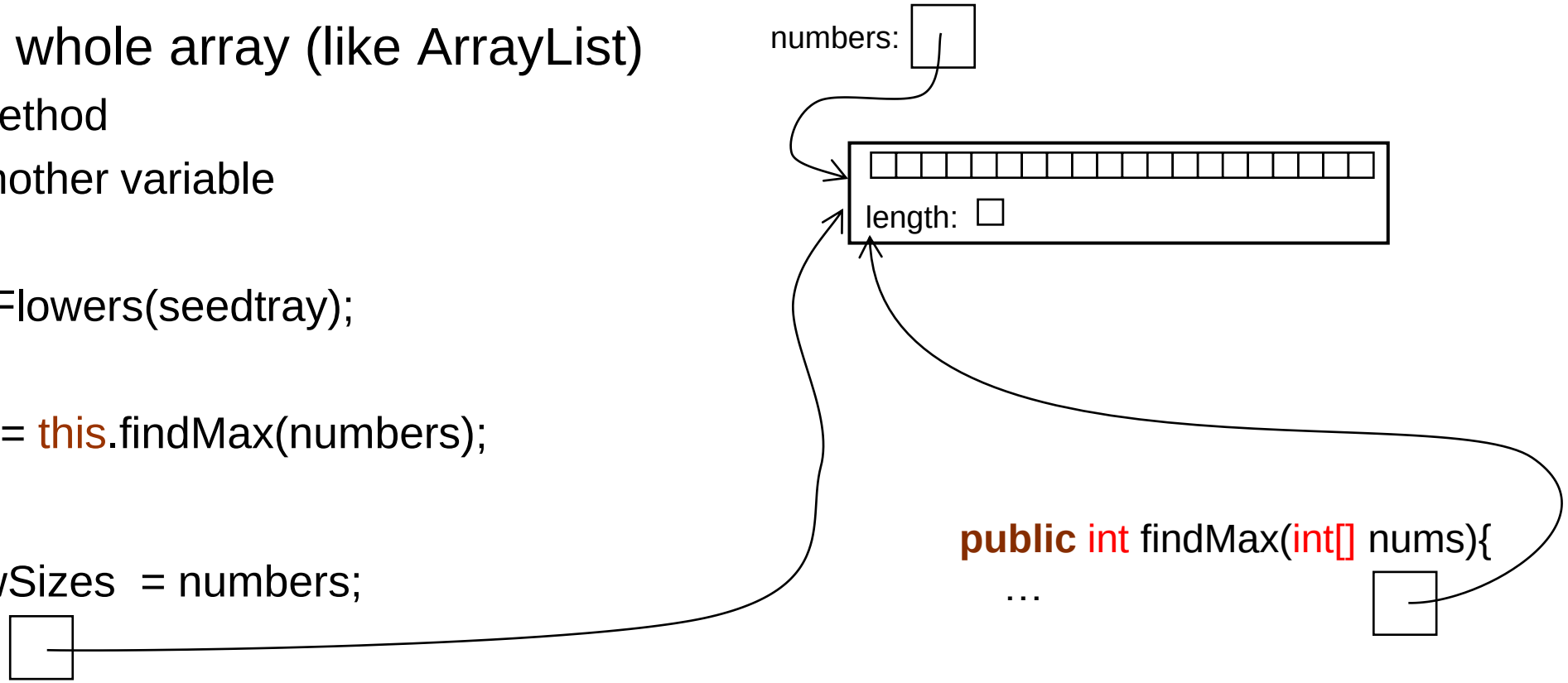
No ‘Array’ in declaration!



Using an array

- Can act on the whole array (like ArrayList)
  - to pass to a method
  - to assign to another variable

```
    :
this.processFlowers(seedtray);

int maxNum = this.findMax(numbers);

int [ ] windowSizes  = numbers;
```



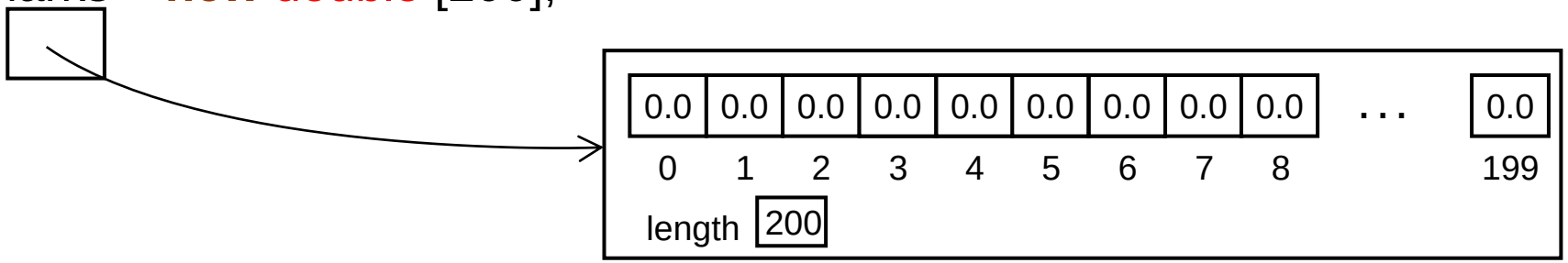
- Note, passing as argument and assignment do **not** copy the array! (just the reference/ID of the object)
- Just the same as with ArrayList.

Using an Array

- Use [..] to refer to an individual place in the array
  - to access the value in that place
  - to put a value in that place (using assignment: =)

Not get() and set()

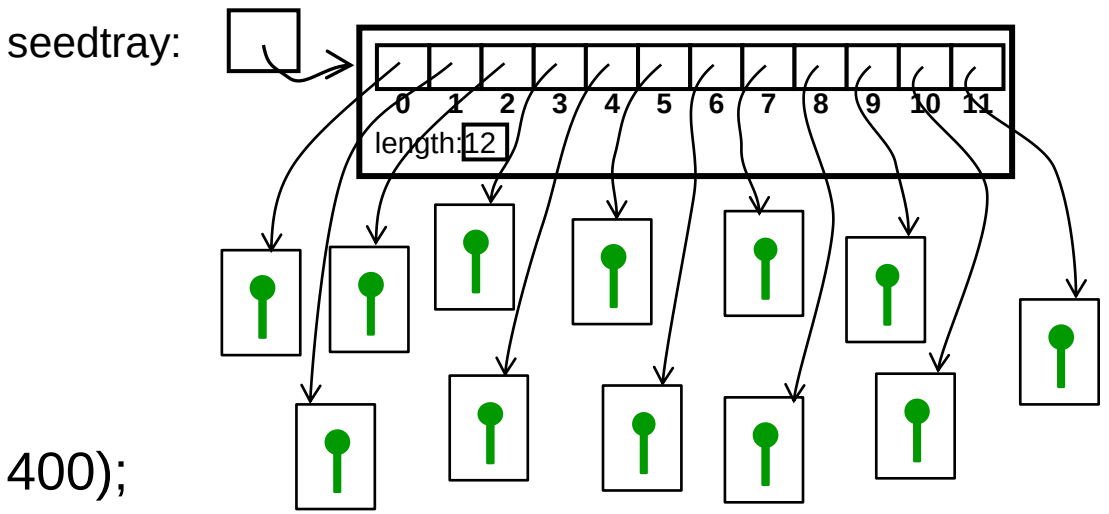
```
double [ ] marks = new double [200];

int n=4;
   :

marks[5] = 45.6;
marks[6] = ( marks[5] + marks[7] ) / 2;
marks[n-1] = 80.0;
marks[n] = marks[n-1];
if (marks[ i ] == marks[ i+1 ]) {…
```



Index can be any int valued expression

SeedTray Program

```
public class  SeedTray{
    private  Flower[ ] seedtray = new  Flower[12];
        :
    public void replant(){
        for (int i = 0; i < this.seedtray.length;  i++) {
            this.seedtray[ i ] = new  Flower(70+i*50, 400);
        }
    }
    public  void growAll(){
        for (int i = 0; i < this.seedtray.length; i++) {
            this.seedtray[ i ].grow();
        }
    }
```



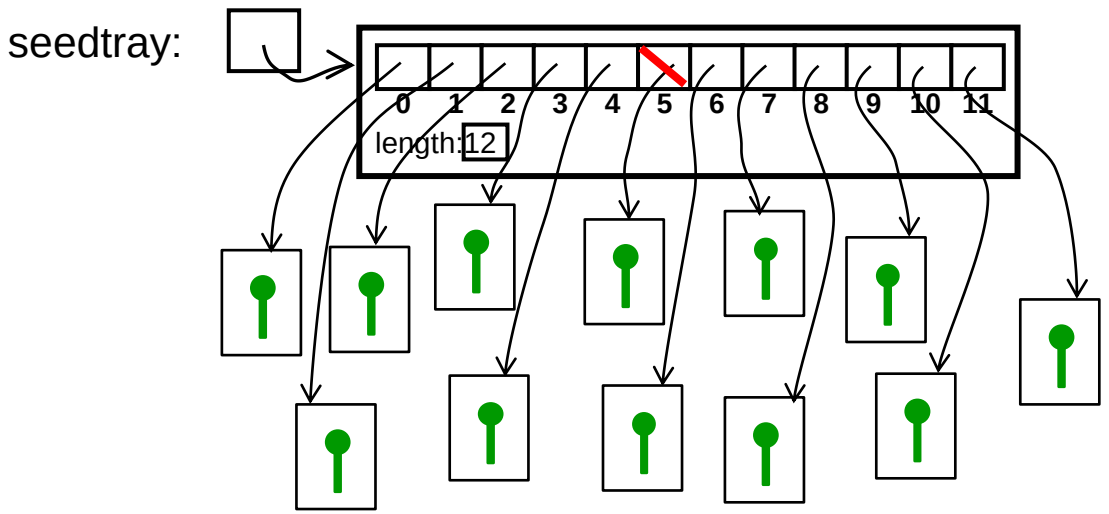

```
    public  void growAll(){
        for (Flower flower : this.seedtray){
            flower.grow();
        }
    }
```

For each loop works on arrays, just like ArrayLists

Arrays of Objects can contain null

```
public void pick(int index){
    this.seedtray[ index ] = null;
}
```

If the array may have null, must check items before acting on them



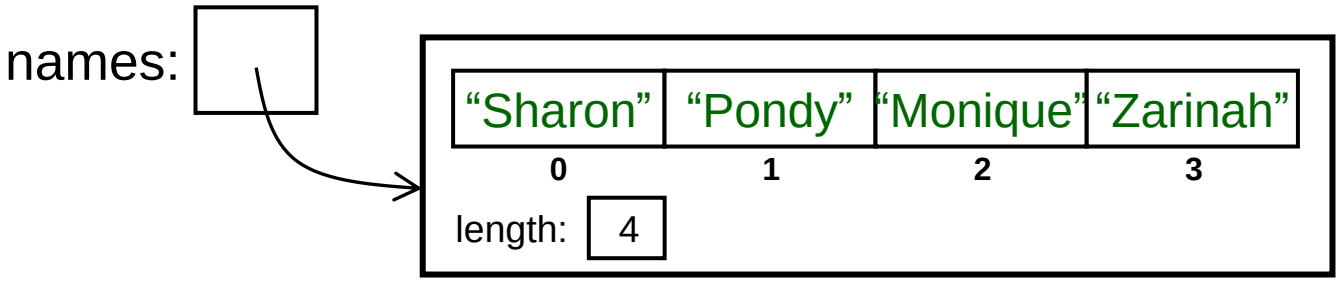

```
public  void growAll(){
    for (int i = 0; i < this.seedtray.length; i++) {
        if (this.seedtray[ i ] != null){
            this.seedtray[ i ].grow();
        }
    }
}
```

```
public  void growAll(){
    for (Flower flower : this.seedtray){
        if (flower != null){
            flower.grow();
        }
    }
}
```

Initialising the contents of an array

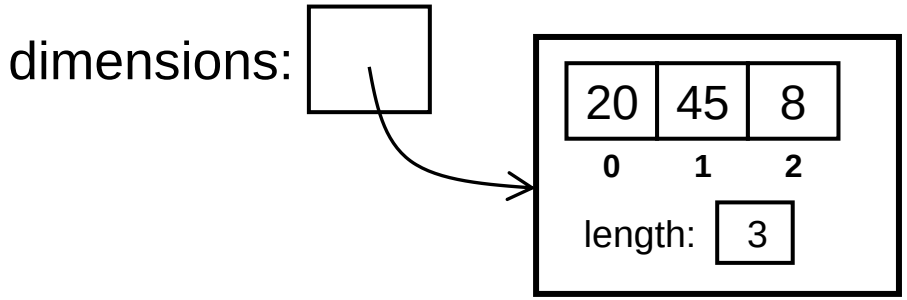
- Can specify the initial values (and size) of an array by listing the values in {.. , .. , ..} :

```
String [ ] names = new String [ ] { "Sharon", "Pondy", "Monique" , "Zarinah"  };
```



Can't do this with ArrayLists!

```
int [ ] dimensions = new int [ ] { 20, 45, 8 };
```



Arrays vs ArrayList

- Use an array if
  - it will never change size, and
  - you know how big it will need to be, at the point you need to create it.

- Use an ArrayList if
  - the size will change, or
  - you don't know how big it will need to be.

- Arrays have convenient syntax [ ]
- ArrayLists have convenient methods.

Saving an Array to a File for later Loading

- Saving & loading an array is similar to saving an ArrayList:
  - Step through the array, writing each value to the file, or reading each value from the file.
  - But: Must be careful with null values.
    Must know the number of items **before** we construct the array.

- Three options:

  a) write one line for each item in the file; write "null" for null values
  read lines into an arraylist, then create array. Check for lines with "null"
  ➔ size of arraylist tells us the size of the array,
  but inefficient for very large files or arrays with lots of null values.

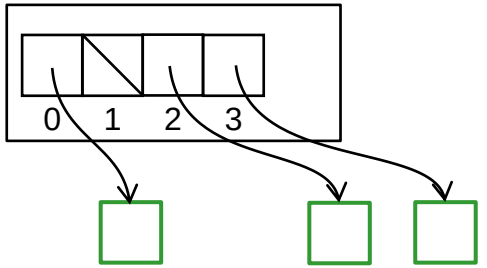
  b) write the size of the array to the file before writing the values.
  ➔ can read the size, then construct array, then read the values

  c) write the size of the array to the file first, then
  for each non-null value, write the index and the value on a line
  ➔ can read the size, then construct array, then read the index – value pairs.

Saving and Loading, option (a)

```
public void save(){
    try{
        String fname = UIFileChooser.save("File for Seedtray");
        PrintStream out = new PrintStream(fname);
        for (Flower flower  :  this.seedtray) {
            if (flower == null) {
                out.println("null");
            }
            else {
                out.println(flower);
            }
        }
        out.close();
    }catch (IOException e) { UI.println("File saving failed: "+e); }
}
```

seedtray:

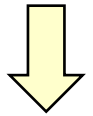



```
104 268  40  bud
null
524 245  20  picked
274  83  50  bud
                :
```

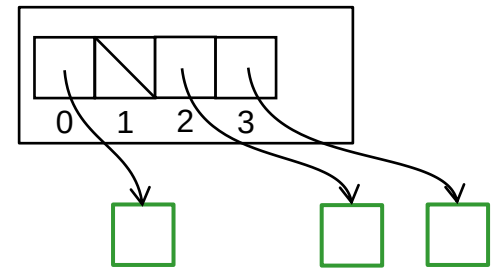
Saving and Loading, option (a)

```
public void load(){
    try{
        String fname = UIFileChooser.open("File containing Seedtray");
        List<String> lines = Files.readAllLines(Path.of(fname));
        this.seedTray = new Flower [ lines.size() ];
        for (int i = 0; i<lines.size(); i++) {
            if ( ! lines.get(i).equals("null") ) {
                Scanner sc = new Scanner(lines.get(i));
                Flower f = new Flower(sc.nextDouble(), sc.nextDouble(), sc.nextDouble(), sc.next());
                this.seedTray [ i ] = f;
            }
        }
    } catch (IOException e) { UI.println("File loading failed: "+e); }
}
```

```
104 268  40  bud
null
524 245  20  picked
274  83  50  bud
          :
```



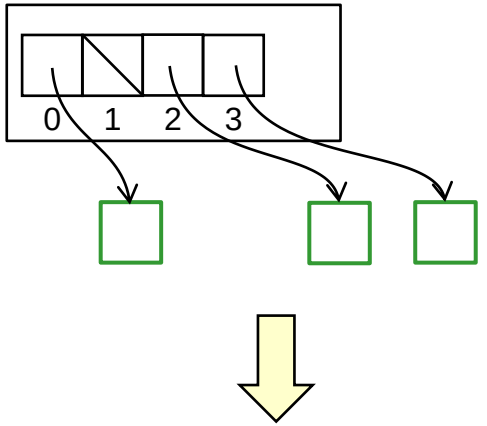
seedtray:



Saving and Loading, option (b)

```
public void save(){
    try{
        String fname = UIFileChooser.save("File for Seedtray");
        PrintStream out = new PrintStream(fname);
        out.println(this.seedTray.length);
        for (Flower flower  :  this.seedtray) {
            if (flower == null) {
                out.println("null");
            }
            else {
                out.println(flower);
            }
        }
        out.close();
    }catch (IOException e) { UI.println("File saving failed: "+e); }
}
```

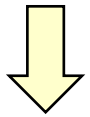
seedtray:




```
4
104 268  40  bud
null
524 245  20  picked
274  83   50  bud
                :
```

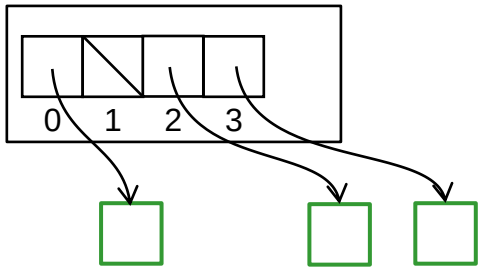
Saving and Loading, option (b)

```
4
104 268  40  bud
null
524 245  20  picked
274  83  50  bud
              :
```



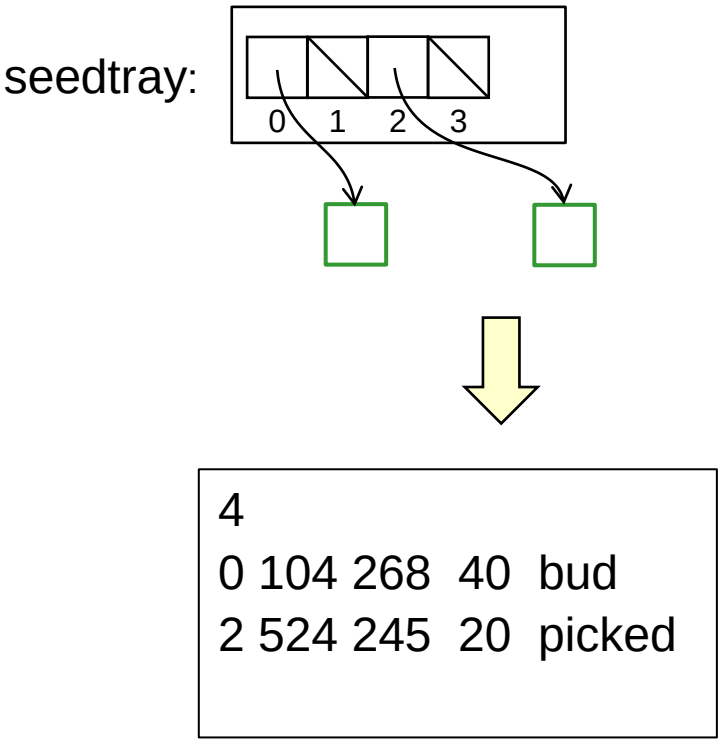
```
public void load(){
    try{
        String fname = UIFileChooser.open("File containing Seedtray");
        Scanner sc = new Scanner(Path.of(fname));
        this.seedTray = new Flower [ sc.nextInt() ];
        for (int i = 0; i< this.seedTray.length; i++) {
            if (sc.hasNextDouble()) {
                Flower f = new Flower(sc.nextDouble(), sc.nextDouble(), sc.nextDouble(), sc.next());
                this.seedTray[ i ] = f;
            }
            else { sc.nextLine(); }
        }
        sc.close();
    }
    catch (IOException e){UI.println("File loading failed: "+e);}
}
```

seedtray:



Saving and Loading, option (c)

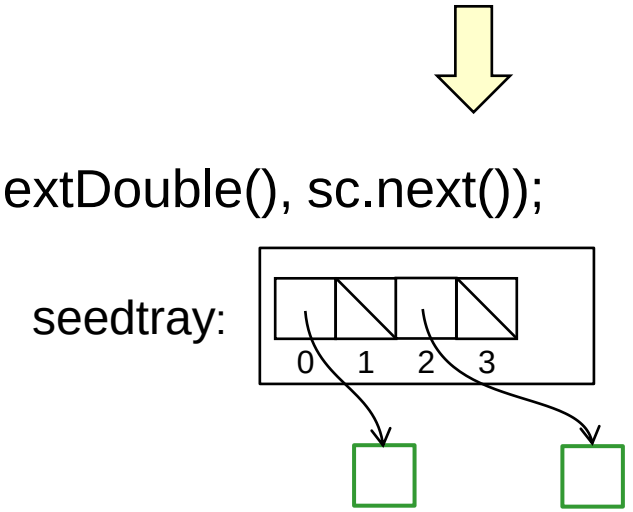
```
public void save(){
    try{
        String fname = UIFileChooser.save("File for Seedtray");
        PrintStream out = new PrintStream(fname);
        out.println(this.seedTray.length);
        for (int i=0; i<this.seedTray.length; i++) {
            if (this.seedTray[ i ] != null) {
                out.println( i + " " + this.seedTray[ i ] );
            }
        }
        out.close();
    }catch (IOException e) { UI.println("File saving failed: "+e); }
}
```



Saving and Loading, option (c)

```
4
0 104 268  40  bud
2 524 245  20  picked
```

```
public void load(){
    try{
        String fname = UIFileChooser.open("File containing Seedtray");
        Scanner sc = new Scanner(Path.of(fname));
        this.seedTray = new Flower [ sc.nextInt() ];
        while ( sc.hasNext() )  {
            int indx = sc.nextInt();
            Flower f = new Flower(sc.nextDouble(), sc.nextDouble(), sc.nextDouble(), sc.next());
            this.seedTray[indx] = f;
            }
        sc.close();
    }
    catch (IOException e){UI.println("File loading failed: "+e);}
}
```

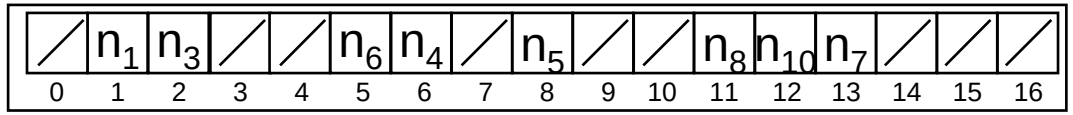


More ways of using arrays

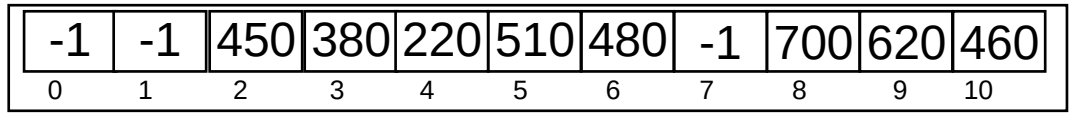
Sometimes, the index represents meaningful information:
More than just the position in the array.

For example:

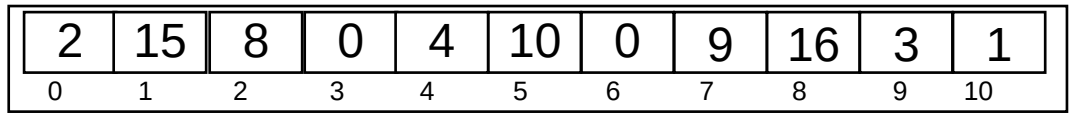
- array of guest names for hotel rooms (numbered 1 to MaxRoom)



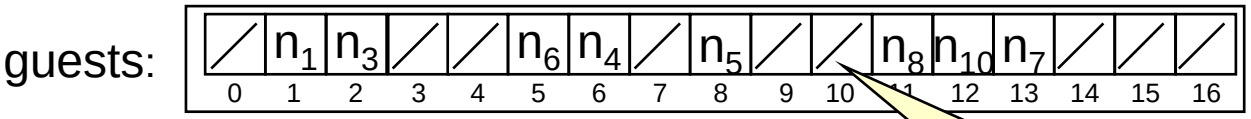
- house info for each house on street



- count of occurrences of each number



Hotel Register



null: the "not-a-real-object" value. Can always be assigned to a place of an object type

- initialise empty

```
private String[ ] guests = new String[MaxRoom+1];
  // gives indexes from 0 to MaxRoom, ignore index 0
```

- assign to room

```
public void assignGuest(String name; int room){
    this.guests[room] = name;
}
```

Don't need to step through array!

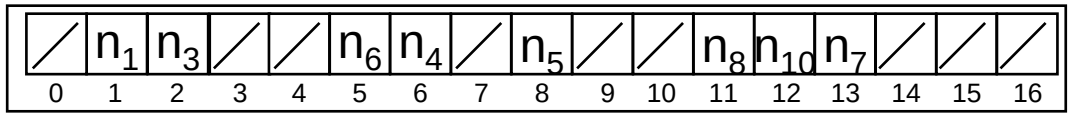
- look up to see if free

```
public boolean isFree(int room){
    return (this.guests[room]==null);
}
```

Hotel Register: remove

Checkout guest from room

guests:




```
/* returns true if name was checked out of room successfully,
   false otherwise */
public boolean checkout(String name, int room){
    if ( this.guests[room].equals(name) ){
        this.guests[room] = null;
        return true;
    }
    return false;
}
```

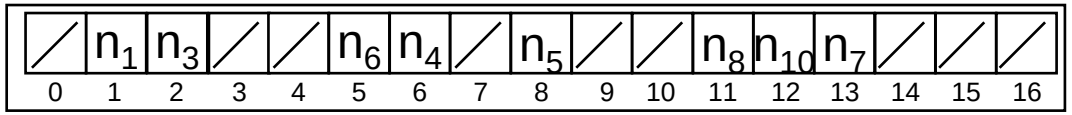
Problem if null!
How do we fix it?

```
if ( this.guests[room] != null  &&  this.guests[room].equals(name) ) …
if ( name.equals(this.guests[room]) ) …
```

Alternative

Checkout guest: search and remove:

guests:




```
/* returns true if name was checked out successfully,
   false otherwise */
public boolean checkout(String name){
    for (int rm=1;   rm<this.guests.length;  rm++){      // or <=MaxRoom
        if (this.guests[rm] != null && this.guests[rm].equals(name) ) {
            this.guests[rm] = null;
            return true;
          }
        }
    }
    return false;
}
```

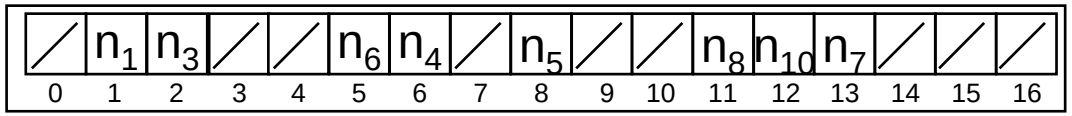
Example of "Any" pattern

Find an empty room

Find the index of an empty room (return -1 if no empty rooms)

```
public int findEmpty(){
    for (int rm=1;  rm<this.guests.length;  rm++){       // or <=MaxRoom
        if (this.guests[rm]==null)   {  return rm;  }
    }
    return -1;
}
```

guests:



Check a guest into an empty room (return room number)

```
public void checkIn(String name){
    this.guests[ this.findEmpty() ] = name;
}

public boolean checkIn(String name){
    int rm = this.findEmpty();
    if (rm < 0) { return false; }
    this.guests[rm] = name;
    return true;
}
```

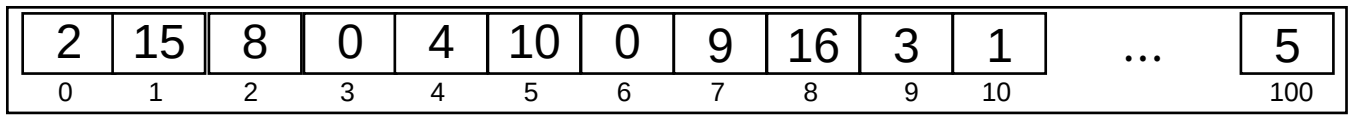
What's the problem?
How do we fix it?

Arrays of Counts

- To keep track of numbers we have seen, efficiently
- Counting the number of occurrences of each number a file of integers:

```
Scanner sc = new Scanner(Path.of(UIFileChooser.open("file to check")));
int[ ] numbersSeen = new int[101];
while ( sc.hasNext() ){
    if ( sc.hasNextInt() ){
        int num = sc.nextInt();
        if ( num>=0 && num<=100 ){
            numbersSeen[num] = numbersSeen[num] + 1;      //  OR     numbersSeen[num]++;
        }
    }
    else{
        sc.next();
    }
}
```

Initialised with 0



Comparing arrays.

- Be careful when comparing arrays (as with all objects)

```
int[ ] a = new int[ ]{ 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 43, 47};
int[ ] b = new int[ ]{ 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 43, 47};
int[ ] c = b;
```

```
if (a == b) ..                          ➔ ??
if (b == c) ..                          ➔ ??
if (a.equals(b) ) ..                    ➔ ??
if  (Arrays.equals(a, b) ) ..           ➔ ??
if  (this.myIntArrayEquals(a, b) ) ..   ➔ ??
```

```
public boolean myIntArrayEquals(int[ ] a, int[ ] b) {
    if (a==null && b==null ) { return true; }
    if (a==null || b==null ) { return false; }
    if ((a.length != b.length ) { return false; }
    for (int i = 0; i < a.length; i++) { if ( a[i] != b[i] ) { return false; } }
    return true;
}
```