



# Algorithmic Thinking and Structured Programming (in Greenfoot)

---

Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

Ana Tanase



# Today's Lesson plan (10)

---

- Retrospective
  - Previous lesson
  
- Theory:
  - Nested if.. then.. else
  - Class constants
  - Object types (and null)
  - Lists and for-loops
  - use Java Library Documentation to look for and use existing Java (List) methods;
  
- Exercises



# Retrospective

---

Steps for using **instance variables**

1. **Declare** instance variable in top of **class**:

```
private boolean iAmHatched;
```

2. **Initialize** (set initial value) in **constructor**:

```
iAmHatched = false;
```

3. Write **public getter accessor** method

```
public boolean getIsHatched (){  
    return iAmHatched;  
}
```

4. Write **public setter mutator** method:






```
public void setHatched( ){  
    iAmHatched = true;  
}
```





# Sit on the most valuable egg

## Make a flowchart

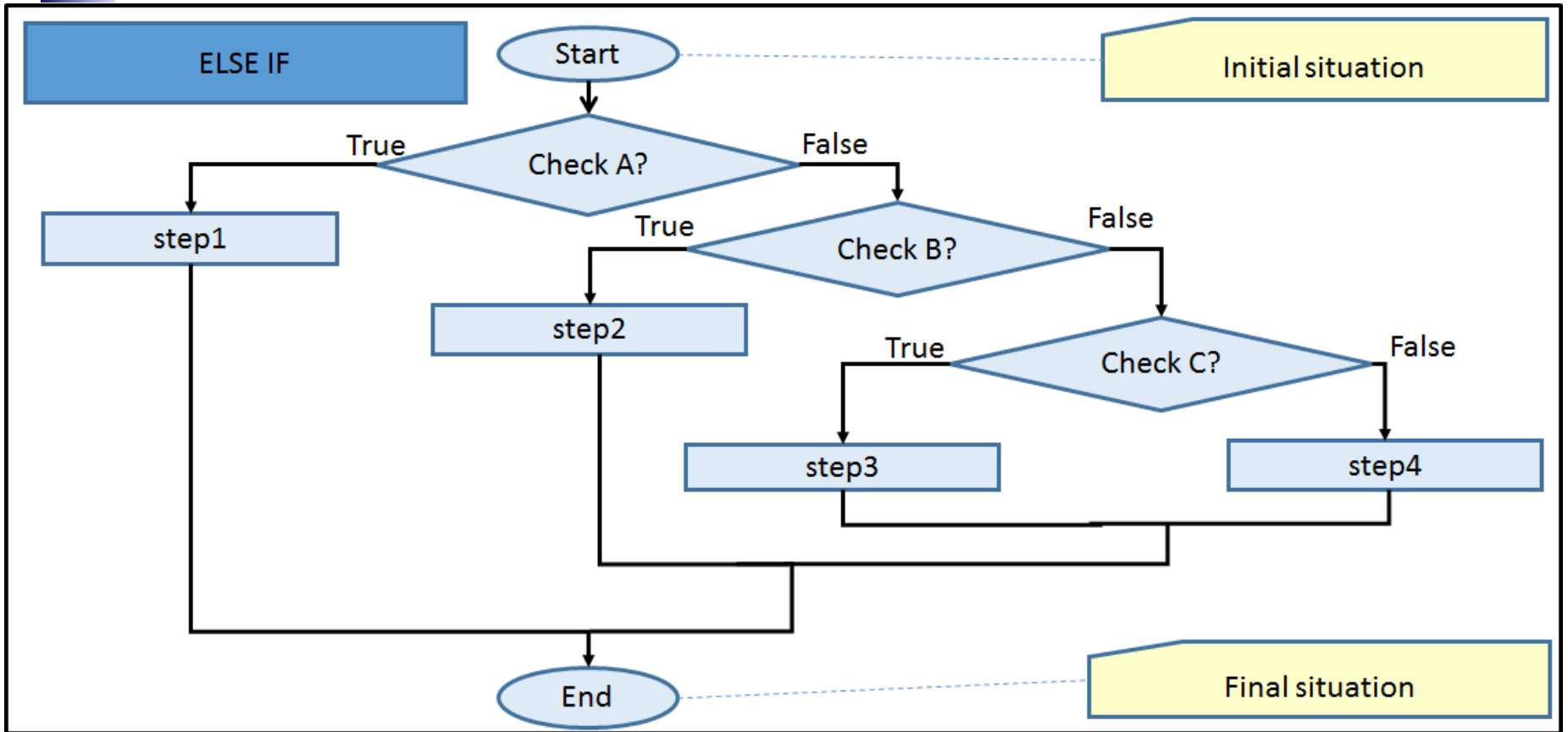
You may assume the following methods exist:

- ❑ boolean **eggOneStepAway**( String color )
- ❑ void **sitOnEggOneStepAway** ( String color )

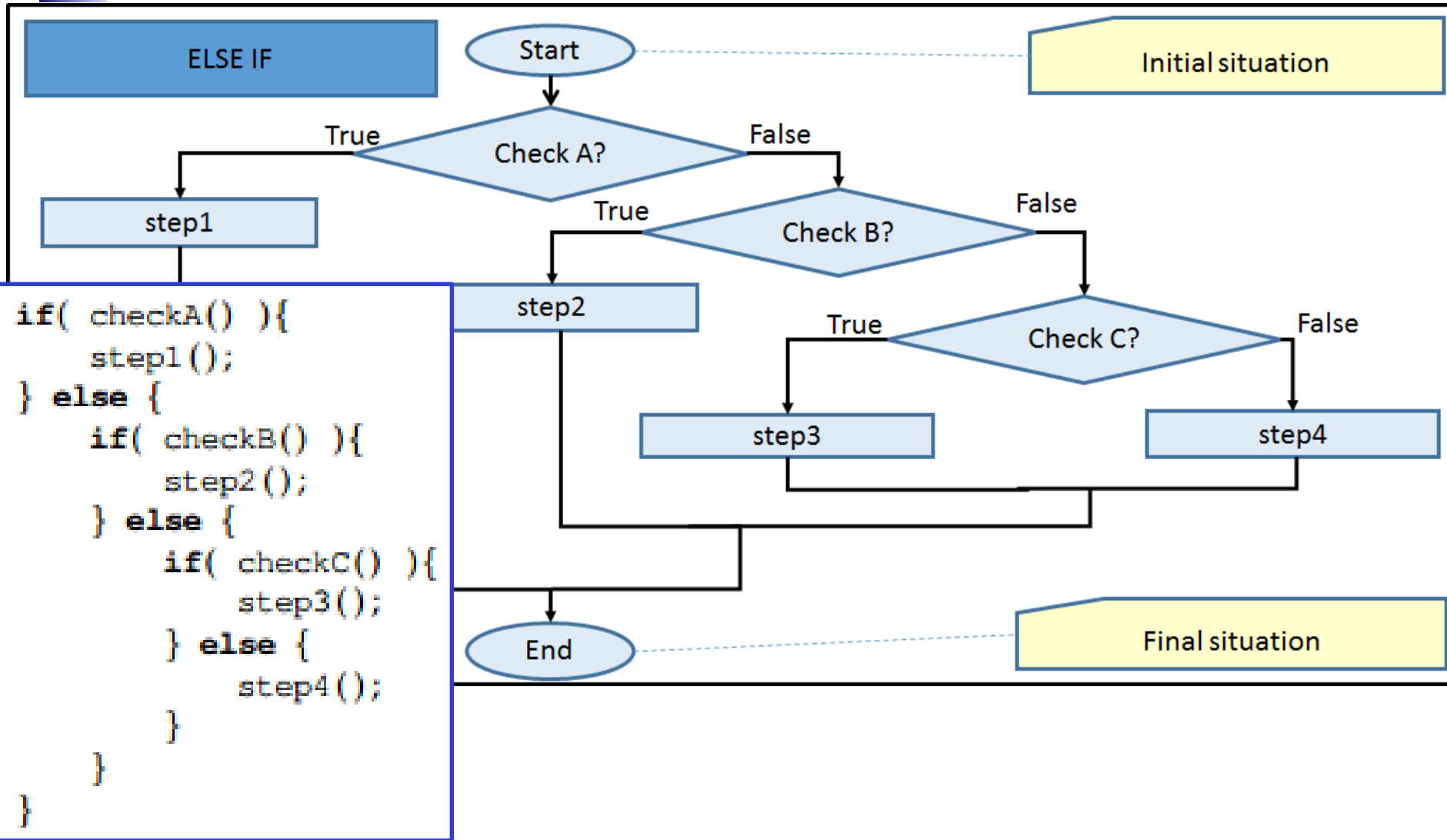
					
					
					

	10 points
	5 points
	2 points
	1 point

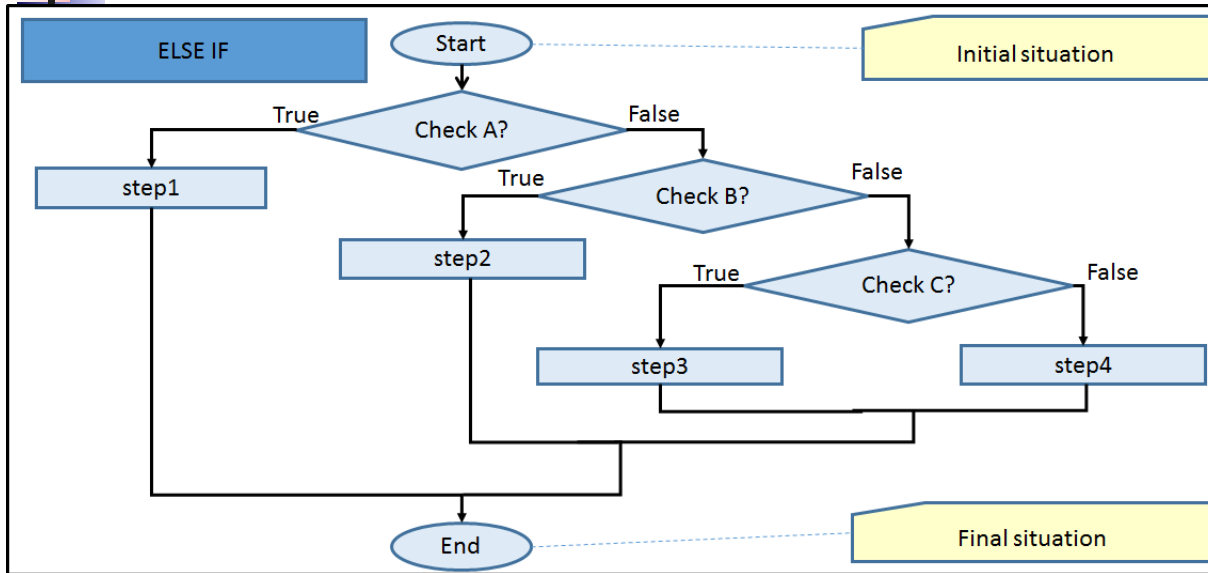
# Nested if-then-else



# Nested if-then-else



# Nested if-then-else => else-if



```
if( checkA() ){
    step1();
} else {
    if( checkB() ){
        step2();
    } else {
        if( checkC() ){
            step3();
        } else {
            step4();
        }
    }
}
```

code simplified

using else if

```
if( checkA() ){
    step1();
} else if ( checkB() ){
    step2();
} else if ( checkC() ){
    step3();
} else {
    step4();
}
```



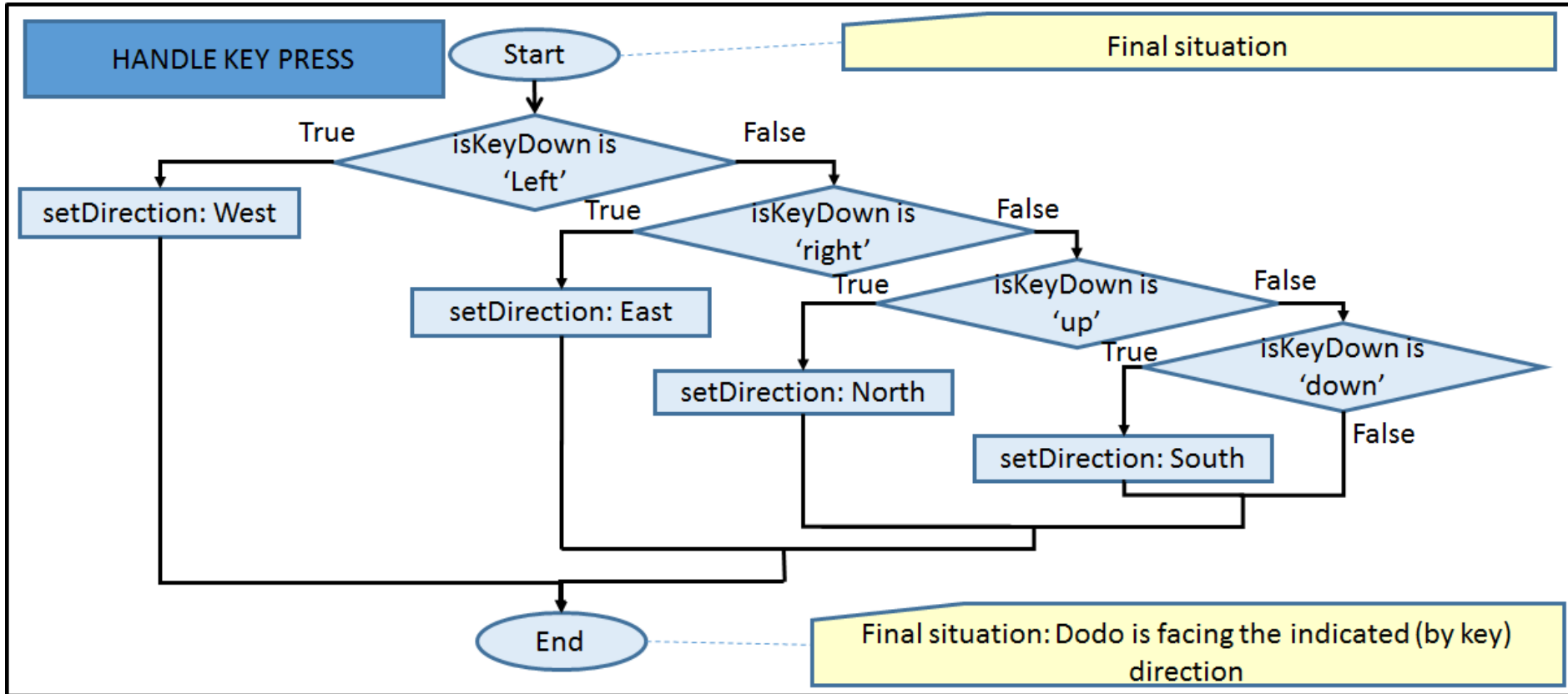
# Computational thinking

---

- Working in a structured manner:
  - Breaking problems down into subproblems
  - Design, solve and test solutions to subproblems
  - Combining these (sub)solutions to solve problem
- **Analyzing** the quality of a solution
- **Reflecting** about the solution chosen and proces
- **Generalizing** and re-use of existing solutions



# Another ex. of nested if-then-else





# Class constants

---

- ❑ Variable whose value **can't change** throughout the **program**
- ❑ Recognized by **static final**
- ❑ When declared, **immediately** give value
- ❑ Can be private or public

## Example

An example of a constant's declaration (at the top of the Madagascar class) is:

```
private static final int MAXWIDTH=12;
```



# Object types vs primitive types

---



# Primitive Datatypes in Java

---

- **Truth values** (booleans)

**boolean:** true and false.

- **Integer values** (integers)

**int:** -1, 0, 42, 123, -51

- **Real values** (reals)

**double:** -1.0, 0.0, 42.0,  
2.1795, 6.02e23, 1.6e-19

- **Characters**

**char:** 'a', 'A', '?', '-', ' ' (= a “space”!)



# Object types: Variables for objects

---

- ❑ Variables can also contain objects
- ❑ More precisely: Object variables **point / refer to** objects
- ❑ The **type** of such a variable is the **class** the object belongs to
- ❑ Such a type is called an **object type** (or **reference type**)
- ❑ Other types (**int**, **boolean**, ..) are called **primitive types**
- ❑ Example:

```
Egg thisEgg = getEgg ();
```

A variable that can hold an Egg object

# Variables as *References*

So, variables can be used to *remember* another object.

- Via such a (*reference*) variable one object can collaborate with (call methods of) another object.

## Example:

In your mobile phone you have a list of *Contacts*.

A contact is a *reference* to a friend, family, ...

*My Contacts*

*Alice*

*Bob*

...



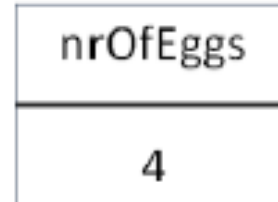
081-555-1212



# Primitive types vs object types

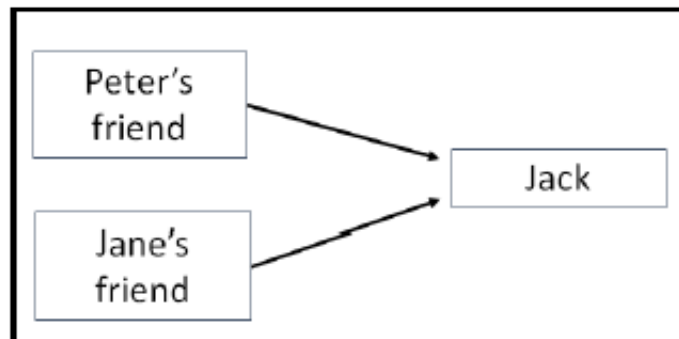
- Primitive type stores value directly in variable:

- Eg. `int nrOfEggs = 4;`



- Object type refer (or points) to another object:

- Eg. Facebook doesn't physically store your friends
- It stores your friends' login names



# Variables containing `null`

- Special value to indicate that a variable does **not refer** to anything:

`null`

- Sometimes methods return this value to say that an **object could not be found**.

```
Egg maybeEgg = getEgg ();
```

- We can use this as follows:

```
Egg maybeEgg = getEgg ();  
if ( maybeEgg != null ) {  
    ...  
}
```

`getEgg` returns `null` if our cell does not contain an egg



# Lists



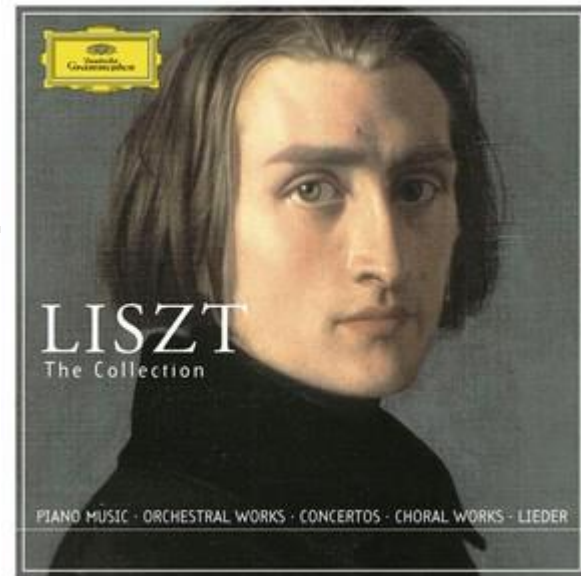
- So far, variables can contain just a single object.

```
Egg thisEgg = getEgg ();
```

A variable that can hold an Egg object

- Sometimes it is convenient to maintain a whole collection of objects
- For this purpose we can use **Lists**. A list can be seen as a sequence of variables: the **elements** of the list.
- A List grows and shrinks to match whatever you put in the list: elements can be added, removed or changed.

# Lists (2)



Properties:

- A list may be **empty**.
- It's a **sequence** → each element can be identified with its position (**index**). The first element has index 0!
- It's **homogeneous**: all the elements are of the same type.

Lists are objects themselves (not a primitive value)

- A variable holding a list object is declared as:
  - `List<ElemType> listVariable;`

The type of each element

# List example: how to use

Create a List of fruit names.

Creates a new empty list

```
public static void listExample(){  
    List<String> fruitList = new ArrayList<String> ();
```

Initial size is 0

```
    System.out.println ( fruitList.size( ) ),
```

Add 3 elements

```
    fruitList.add( "apple" );  
    fruitList.add( "orange" );  
    fruitList.add( "banana" );
```

Size should be 3 now

```
    System.out.println ( fruitList.size( )
```

Prints apple

```
    System.out.println ( fruitList.get(0) ),
```

Prints banana

```
    System.out.println ( fruitList.get(2) )
```

Remove apple from the list

```
    fruitList.remove( "apple" );
```

Prints orange

```
    System.out.println ( fruitList.get(0) )
```

```
}
```



# List example: homogeneous types

Create a List of fruit names (Strings).

```
public static void listExample(){  
    List<String> fruitList = new ArrayList<String> ();  
  
    fruitList.add( "apple" );  
    fruitList.add( "orange" );  
    fruitList.add( "banana" );  
  
    fruitList.add( 13 );  
    fruitList.add( "broccoli" );  
    fruitList.add( "13" );  
  
}
```

Illegal: 13 is not a String

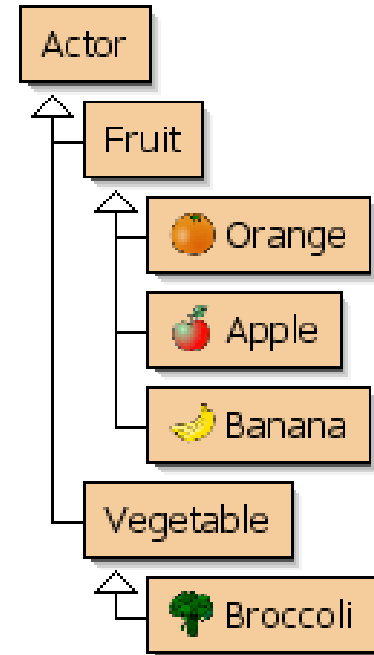
OK: "broccoli" is a String

OK: "13" is a String

# List of objects

Create a List of fruit names (Strings).

```
public static void listExample(){  
    List<Fruit> fruit = new ArrayList<Fruit> ();  
  
    fruit.add( new Apple() );  
    fruit.add( new Orange() );  
    fruit.add( new Banana() );  
  
    fruit.add( new Broccoli() );  
  
}
```



Now we have a list of Fruit elements

OK: Apple 'is a' Fruit

Illegal: Broccoli is no Fruit



# Useful List Methods

<code>list.size( )</code>	Number of items in list.
<code>list.isEmpty( )</code>	<b>true</b> if the list is empty. Same as "list.size() == 0"
<code>list.get( k )</code>	Get one element from list. k = 0, 1, ..., list.size()-1
<code>list.add( object )</code>	Append (add) object to the end of the list.
<code>list.remove( object )</code>	Remove object from a list



# Lists: Examining elements

Using a while loop:

```
// count eggs that are hatched
```

```
List<Egg> eggList = getListOfEggsInWorld();
```

```
int nextEggIndex = 0;
```

```
int nrOfHatchedEggs = 0;
```

Method from class Dodo

```
while( nextEggIndex < eggList.size() ) {
```

```
    Egg egg = eggList.get( nextEggIndex );
```

```
    if ( egg.isHatched() ) {
```

```
        nrOfHatchedEggs ++;
```

Variable holding an index

```
    }
```

```
    nextEggIndex++;
```

Variable for counting

```
}
```



# Lists: what do you need to know

---

- ❑ You don't need to know how to create a list
- ❑ You do need to know how to manipulate and use lists





# Intermezzo

---

- Continue working on assignments
- Finish assignment 6
- Assignment 7:
  - Assignment 7 **up to and incl 4.3.1**
  - You may **skip 4.1 9d and 4.1 10**
  
- After the intermezzo follows: Java Documentation



# The *for each* loop

---

*for each*: a loop for examining all elements of a List (recommended).

```
List<Egg> eggList    = getListOfEggsInWorld();
int nrOfHatchedEggs = 0;

for ( Egg egg: eggList ) {
    if ( egg.isHatched( ) ) {
        nrOfHatchedEggs++;
    }
}
```



"for each egg in eggList"



# While vs for each loop

---

```
List<Egg> eggList = getListOfEggsInWorld();
int nextEggIndex = 0;
int nrOfHatchedEggs = 0;

while( nextEggIndex < eggList.size() ) {
    Egg egg = eggList.get( nextEggIndex );
    if ( egg.isHatched() ) {
        nrOfHatchedEggs ++;
    }
    nextEggIndex++;
}
```

```
List<Egg> eggList = getListOfEggsInWorld();
int nrOfHatchedEggs = 0;

for ( Egg egg: eggList ) {
    if ( egg.isHatched( ) ) {
        nrOfHatchedEggs++;
    }
}
```



# Java documentation

---

- How to find

- Google: “list is empty java”
- look for Oracle Documentation

List (Java Platform SE 7 ) - Oracle Documentation

<https://docs.oracle.com/javase/7/docs/.../java/.../List.ht...> ▼ Vertaal deze pagina

- How to read

- Scroll down and find relevant method

```
boolean
```

```
isEmpty()
```

```
Returns true if this list contains no elements.
```

- How to use

- Click on method name

```
List<Egg> eggList = getListOfEggsInWorld( );  
if ( eggList.isEmpty() ){  
    ...  
}
```

# Java Library Documentation

- We make a list of eggs: `List<Egg> eggList;`
- Get the second element (at index 3) in the `eggList` using: `eggList.get ( 3 );`

□ Returns element of type `<Egg>`

□ Index must be within then list bounds

```
get
E get(int index)
Returns the element at the specified position in this list.
Parameters:
    index - index of the element to return
Returns:
    the element at the specified position in this list
Throws:
    IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())
```



# Computational thinking

---

- **Working in a structured manner:**
  - Breaking problems down into subproblems
  - Design, solve and test solutions to subproblems
  - Combing these (sub)solutions to solve problem
- **Analyzing** the quality of a solution
- **Reflecting** about the solution chosen and proces
- **Generalizing** and re-use of existing solutions



# Wrapping up

---

Homework for Wednesday 8:30 March 2nd:

□ Assignment 7:

- Assignment 7 **up to and incl 4.3.1**
- You may **skip 4.1 9d and 4.1 10**
- **email** MyDodo.java and 'IN'  
to **Renske.weeda@gmail.com**