Algorithmic Thinking and Structured Programming (in Greenfoot)

Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

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, 0.5, 42.0, 2.1795, 6.02e23, 1.6e-19`

- Characters

  `char: 'a','A','?','-', '3', ' '` (= 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

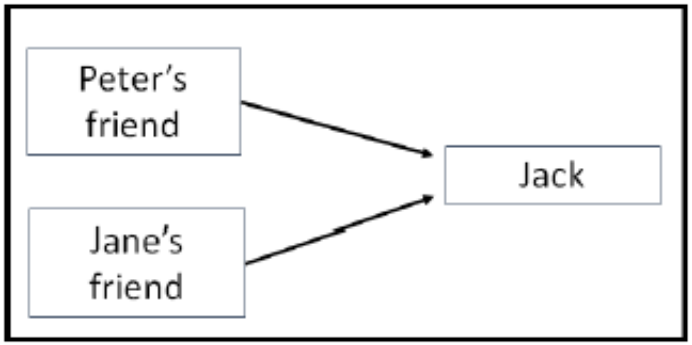
...

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:
  - E.g., 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

Variables as References (2)

Example:

Mimi wants to know "*how big is the world?*"

Each Actor has an instance variable world and a getter method `getWorld( )`.

1. Mimi gets a reference to her world.

```
World myworld = getWorld( ); // in Mydodo
```

Variables as References (3)

2. Now Mimi can *ask the World* some questions, using her *reference* to the world.

```
World myworld = getWorld( );
```

```
int width = myworld.getWidth( );
int height = myworld.getHeight( );
```

Save the answer as (local) variable `height`.
`int` variables contain *values*, not *references*.

Lists

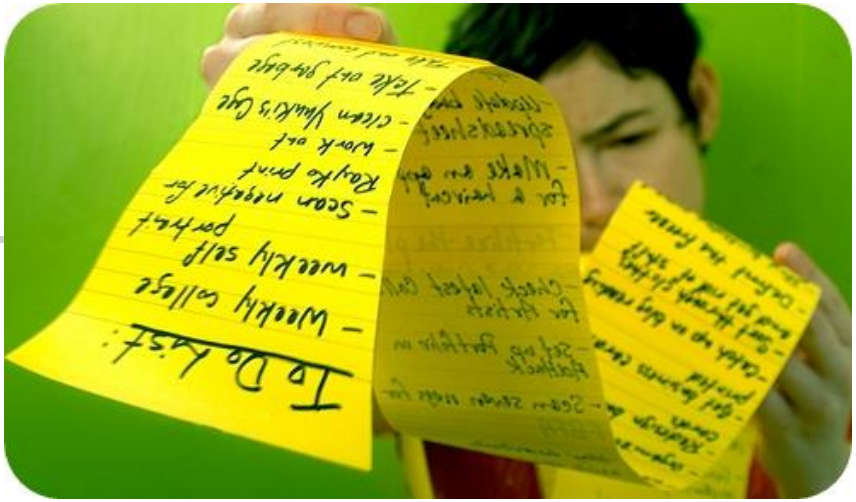
- So far, variables can contain just a single object.

```
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 it's position (index). The first element has index 0!
- It's homogeneous: all the elements are of the same type.

Lists are objects themselves

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.

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

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

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

    System.out.println  ( fruitList.size( )
    System.out.println  ( fruitList.get(0) );
    System.out.println  ( fruitList.get(2) )

    fruitList.remove( "apple" );

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

Local variable holding the list

Creates a new empty list

Initial size is 0

Add 3 elements

Size should be 3 now

Prints apple

Prints banana

Remove apple from the list

Prints orange

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

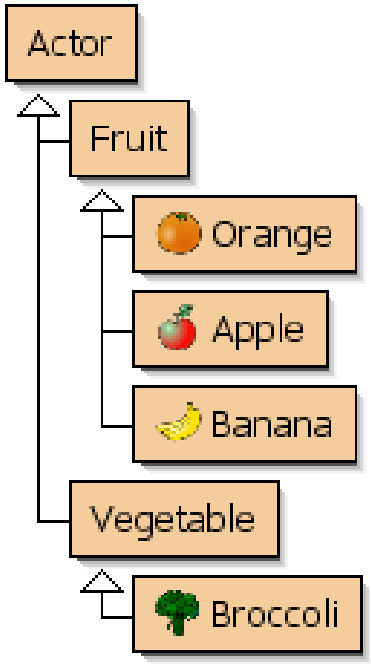
Create a List of fruit names (Strings).

```
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

`list.size( )`	Number of items in list.
`list.isEmpty( )`	**true** if the list is empty. Same as "list.size() == 0"
`list.get( k )`	Get one element from list. k = 0, 1, ..., list.size()-1
`list.add( object )`	Append (add) object to the end of the list.
`list.remove( object )`	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;

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

Method from class Dodo

Variable holding an index

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

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++;
        }
}
```

Dodo's race (goal)

Who can make Dodo the smartest?

- Competition in class on March 18th
  - everyone's program will be run!

Highest score in

max 40 moves WINS!


1 point

5 points

Dodo’s race (rules)

Ground rules:

- Maximum steps: 40
-  15 blue eggs: each worth 1 point
-  1 Golden Egg: worth 5 points
- Mimi only moves using move()
- Max 1 move() per act()
- Competition will be held in a new world
- Highest score wins

Presentation: May 25th

- Presentation:
  - Present (describe) your algorithm to the class (2 minutes)
  - Test your algorithm against classmates

- Who will make the smartest Dodo?
- Think about efficiency (vs brute force)!

Wrapping up

Homework for Wednesday 8:30 May 25th:

- Assignment 7:
  - **FINISH assignment 7 (incl Dodo's race)**
  - email Java code and 'IN'-answers to **sjaaksm@live.com**