Some slides contain lots of animation; you must be in class to fully understand them.

Arrays

covering

** why do we need arrays
** declaring, using and initialising arrays
** copying and printing arrays
** using methods with arrays
** Javadocs

Chapter 6 – “Big Java” book
Chapter 3 – “Head First Java” book
Chapter 5 – “Introduction to Java Programming” book
Chapter 2 – “Java in a Nutshell” book

Arrays & How to declare an array

- An array is a type of data structure like … maps, trees, and sets.
  - Complicated … we will cover this in Week 4.

- Arrays provide fast random access by letting you use an index position to get any element in the array.
  - Arrays must be given a size!
  - Determining the size of an array … more about this soon.

- Two approaches are allowed:

Variable declaration
`typeName arrayRef[];` `typeName[] arrayRef;`

Preferred style

```
String[] args;
```

OR

```
String args[];
```

```
double[] stuff;
```

OR

```
double stuff[];
```

An array of `int` values

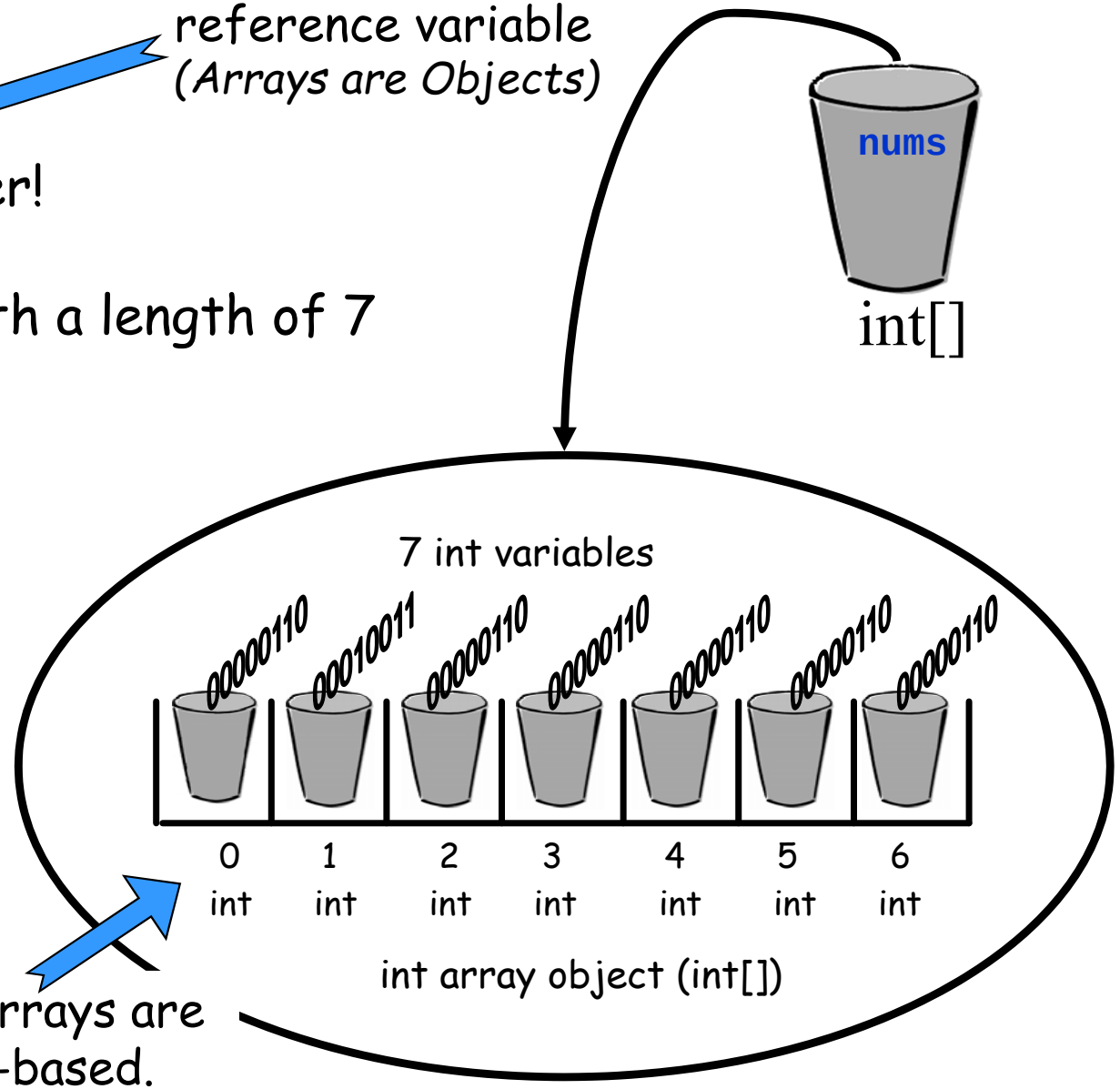
1. Declare int array:
   `int[] nums;`
   acts like the controller!

reference variable
(Arrays are Objects)

2. Create new array with a length of 7
   `nums = new int[7];`

3. Give each element in the array a value.

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```



Arrays are 0-based.

Things to remember about arrays …

- An array is an object, even though it may be an array of primitives!

- Array elements can be either primitives or objects.

- To access things in an array:

```
arrayName[index]
nums[4] = 7;
nums[8] = 9; // causes ArrayIndexOutOfBounds exception
```

More about this later …

Remember the method `main()`?

- It has an array too!
- In `main()`, we have:

Array of `String`s

```
public static void main(String[] args) {
  // print out first array value of args
  System.out.println(args[0]); // args[i]
}
```

- How do we know how many arguments were passed in?

```
public static void main(String[] args) {
  System.out.println(args.length);

  // print out all valid args values
  for (int i=0; i<args.length; i++)
    System.out.println(args[i]);
}
```

… and things for you to try out!

Which is better (or more reusable)?

```
double[] array = new double[6];
// code to fill array ...
for (int i=0; i<6; i++) {
  System.out.println(array[i]);
}
```

Change the array size to **12**.

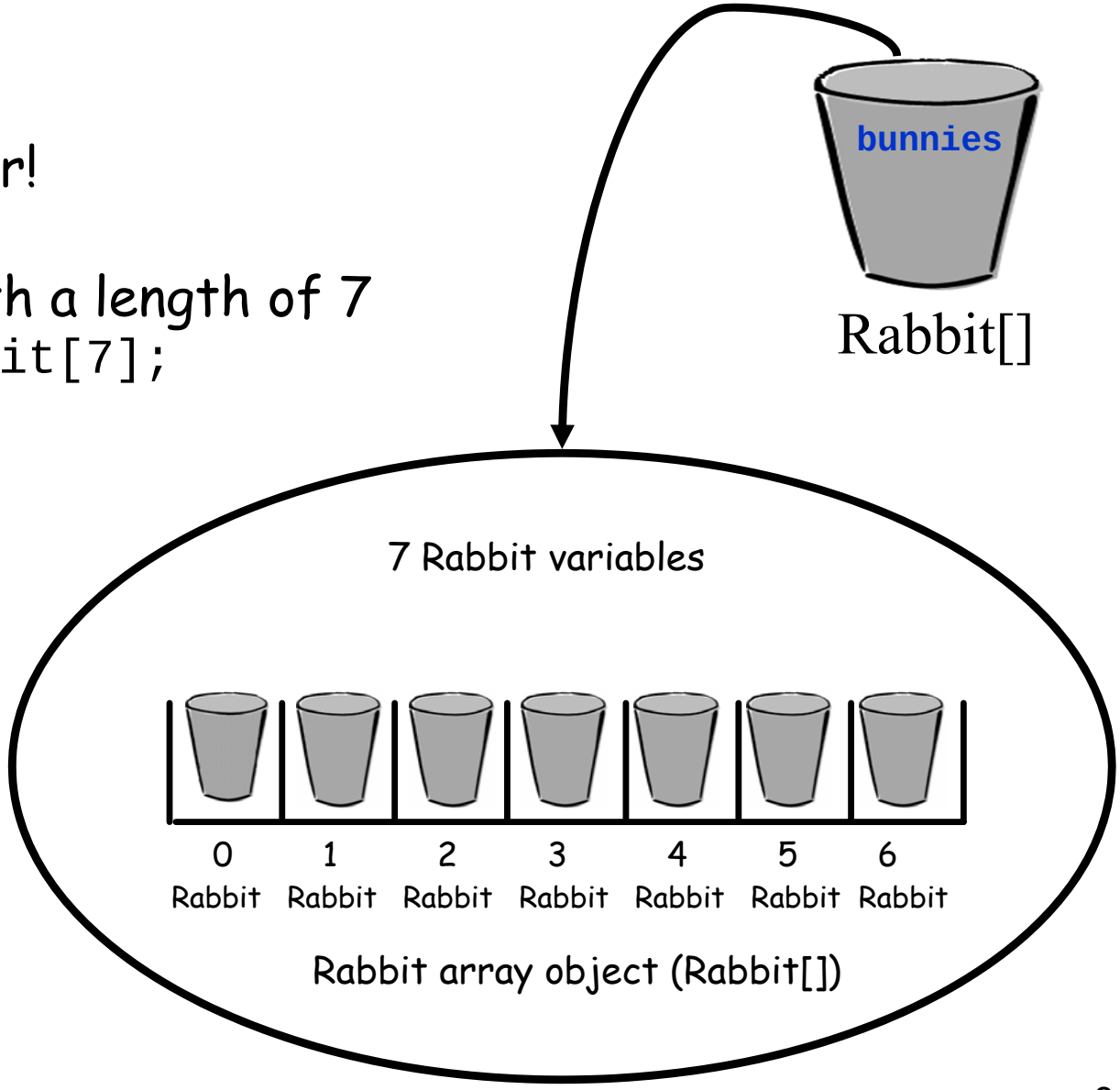
'Hard-coded' array sizes will require modifications throughout the code!

```
double[] array = new double[6];
// code to fill array ...
for (int i=0; i<array.length; i++) {
  System.out.println(array[i]);
}
```

Arrays of objects: A Rabbit array

1. Declare Rabbit array:
   `Rabbit[] bunnies;`
   acts like the controller!

2. Create new array with a length of 7
   `bunnies = new Rabbit[7];`



A Rabbit array

What are we missing?

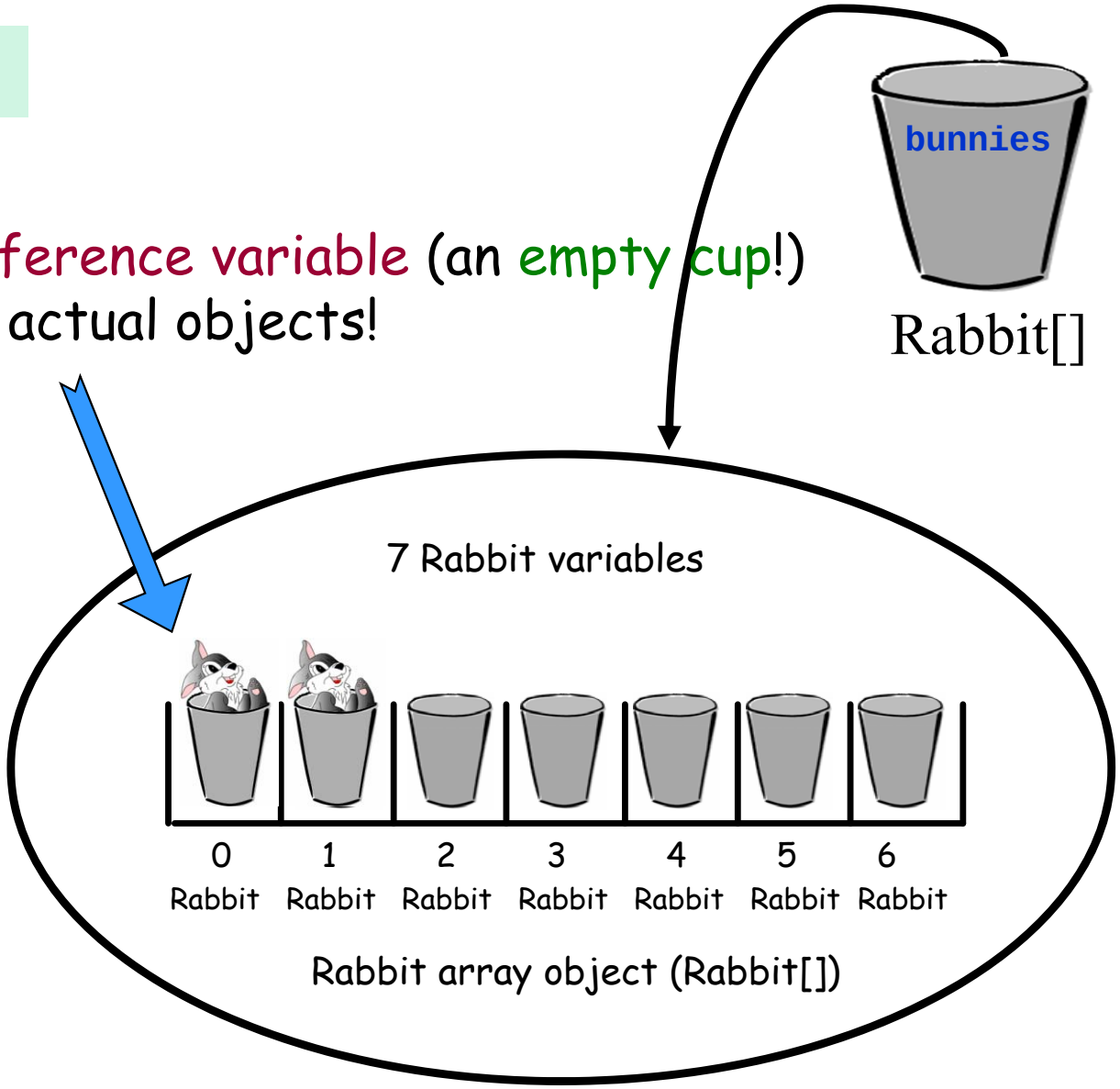
Remember:

- each element is a reference variable (an empty cup!)
- need to assign some actual objects!

Give each element in the array a value.

```
bunnies[0] = new Rabbit();

bunnies[1] = new Rabbit();
```



Using the `Rabbit` class

- We know we can access **`Rabbit`** (**`public`**) instance variables and methods using the dot operator!

```
Rabbit r = new Rabbit();
r.setName("Bugs");
r.setFurType("Fluffy");
r.sleep(5);
```

How do we do that on an array?

```
Rabbit[] racers = new Rabbit[2];
racers[0] = new Rabbit();
racers[0].setName("Bugs");
racers[0].setFurType("Fluffy");
racers[0].sleep(5);
```

Creating `Rabbit` objects in a test class …

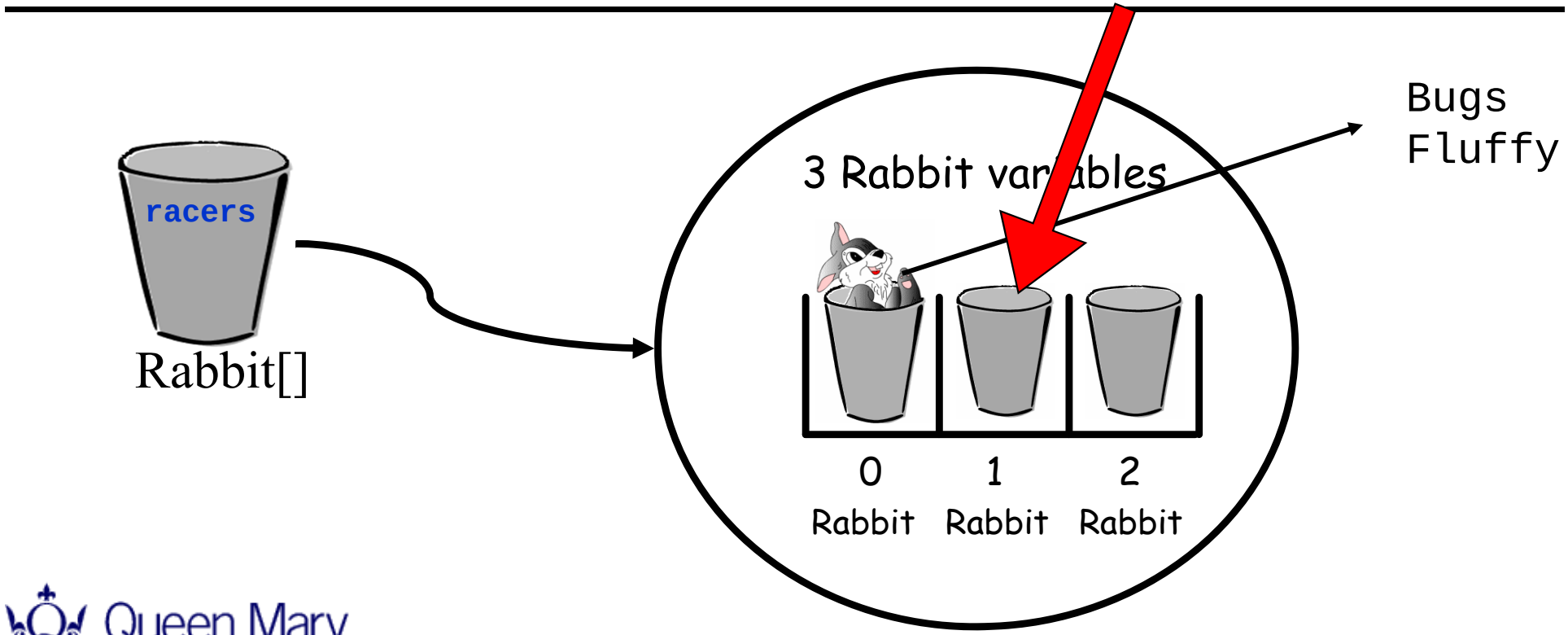
```
Rabbit[] racers = new Rabbit[3];
racers[0] = new Rabbit();
racers[0].setName("Bugs");
racers[0].setFurType("Fluffy");

racers[1].setName("Bunny");
```

`NullPointerException`!

Why?

No actual **`Rabbit`** in the cup!



… and things for you to try out!

Another Rabbit test (1/3)

```
public class RabbitTest {
  public static void main(String[] args) {
    Rabbit[] racers = new Rabbit[2];
    racers[0] = new Rabbit();
    racers[0].setName("Bugs");
    racers[0].setFurType("Fluffy");
    racers[0].setSpeed(150);

    racers[1] = new Rabbit();
    racers[1].setName("Bunny");
    racers[1].setFurType("Long-haired");
    racers[1].setSpeed(145);

    racers[2] = new Rabbit();
    racers[2].setName("Bob");
    racers[2].setFurType("Shaggy");
    racers[2].setSpeed(120);

    for (int i = 0; i < racers.length; i++) {
      System.out.println(racers[i].getName() + " is a " +
        racers[i].getFurType() + " Rabbit that runs at " +
        racers[i].getSpeed() + " km/hr.");
    }
  }
}
```

Running the code throws this exception to the command line: **ArrayIndexOutOfBoundsException***

```
* Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException: 2
  at rabbits.RabbitTest.main(RabbitTest.java:24)
```

Queen Mary University of London

Another Rabbit test (2/3)

```
public class RabbitTest {
  public static void main(String[] args) {
    Rabbit[] racers = new Rabbit[3];
    racers[0] = new Rabbit();
    racers[0].setName("Bugs");
    racers[0].setFurType("Fluffy");
    racers[0].setSpeed(150);

    racers[1] = new Rabbit();
    racers[1].setName("Bunny");
    racers[1].setFurType("Long-haired");
    racers[1].setSpeed(145);

    racers[2] = new Rabbit();
    racers[2].setName("Bob");
    racers[2].setFurType("Shaggy");
    racers[2].setSpeed(120);

    for (int i = 0; i < racers.length; i++) {
      System.out.println(racers[i].getName() + " is a " +
        racers[i].getFurType() + " Rabbit that runs at " +
        racers[i].getSpeed() + " km/hr.");
    }
  }
}
```

Correct now!

No array location for this **Rabbit**!

Output?

Another `Rabbit` test (3/3)

- What if we wrote another program using the **`Rabbit`** class?
  - When we tried to print out the **`Rabbit`** objects, we would have to write the same **`println()`** statement!

This is not object-oriented!

We should always aim to only write code that does a specific thing once!

… and things for you to try out!

Printing arrays and the `toString()` method

- What about:

```
for (int i = 0; i < racers.length; i++) {
  System.out.println(racers[i]);
}
```

Output?

```
Rabbit@3e25a5
Rabbit@923e30
Rabbit@9cab16
```

Not what we are looking for!

- Using the **`toString()`** method will work!

```
public class Rabbit {
  // all the stuff from Rabbit ... previous lectures
  public String toString() {
    return this.getName() + " is a " + this.getFurType() +
    " Rabbit that runs at " + this.getSpeed() + " km/hr.";
  }
}
```

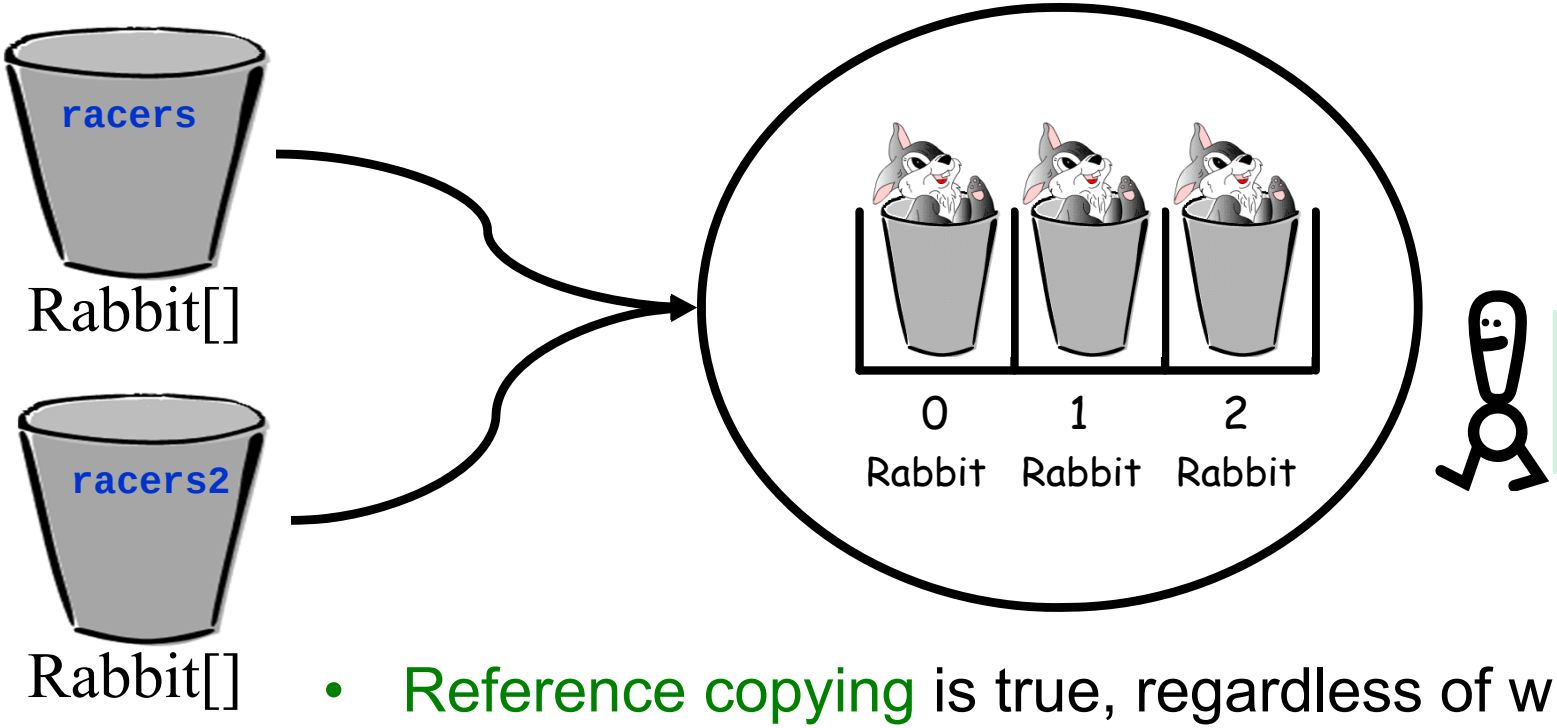
Then the output is:

```
Bugs is a Fluffy Rabbit that runs at 150 km/hr.
Bunny is a Long-haired Rabbit that runs at 145 km/hr.
Bob is a Shaggy Rabbit that runs at 125 km/hr.
```

Copying arrays …

```
Rabbit[] racers2;
racers2 = racers;
```

The result may not be what you expected …



Only copies the reference value!

- Reference copying is true, regardless of whether your array contains primitive or reference variables.
- THREE ways copy the data type variables:
  1. Use a loop to copy all individual elements.
  2. Use the static **`arraycopy`** method in the **`System`** class.
  3. Use the **`clone`** method to copy arrays. (Out of scope)

1. Using a loop to copy elements

```
// nums array – see slide 3 for example

int[] nums2 = new int[nums.length];
for (int i=0; i<nums.length; i++) {
  nums2[i] = nums[i];
}
```

2. Using `arraycopy()` to copy elements

- From the **`System`** class:

Modifier and Type	Method and Description
`static void`	`arraycopy(Object src, int srcPos, Object dest, int destPos, int length)` Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

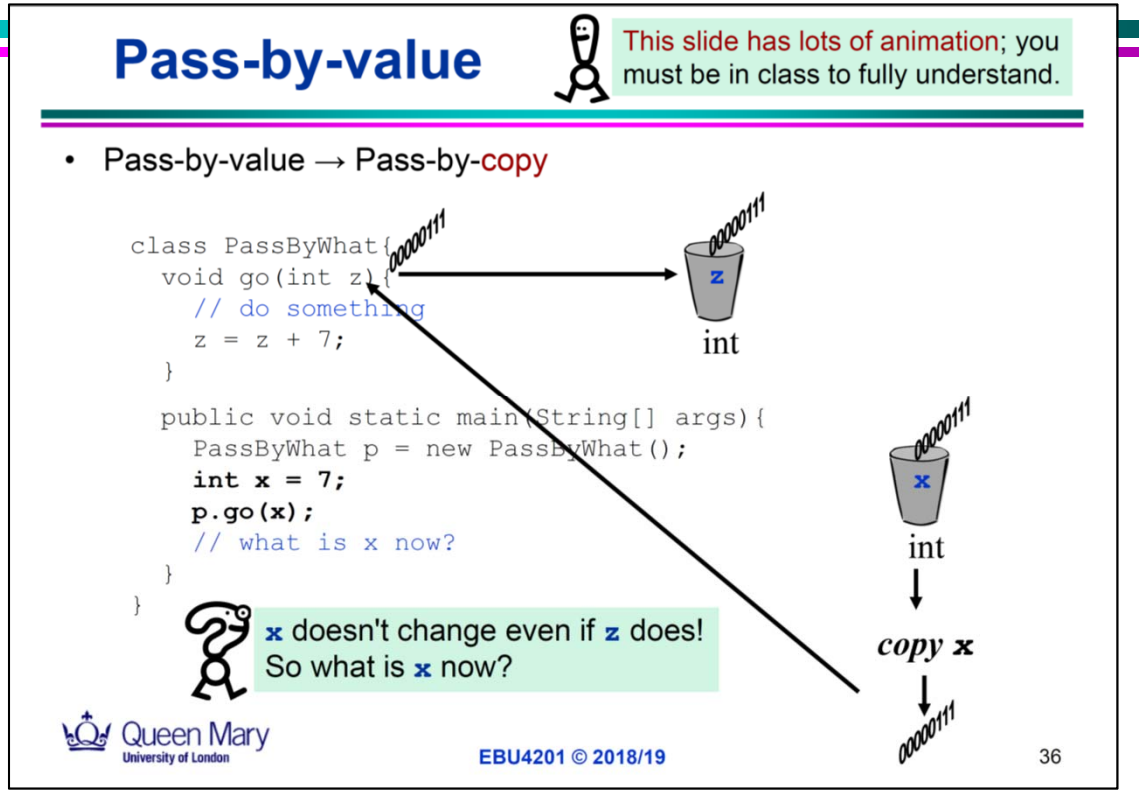
- **`src`** is the array to copy from; **`dest`** is the array to copy to
- **`srcPos`** is where in the **`src`** array to start copying from.
- **`destPos`** is where, in the **`dest`** array, to start putting the newly copied elements

```
System.arraycopy(nums, 0, nums2, 0, nums.length);
```

Same effect as the loop on previous slide!

Passing Arrays to Methods

- Remember?
  - Java uses pass-by-value, to pass arguments to a method.



- With arrays, we have:
  - Java passes (to the method) a copy of the reference variable to the array object.

Read also the information in QMplus, under *Teaching Week 2 > Articles (with illustrative examples) about "pass-by-value".*

Example: passing an array to a method

```
public class PassByTest {
  public void printArray(double[] array) {
    System.out.print("[ ");
    for (int i=0; i<array.length; i++)
      System.out.print(array[i] + " ");
    System.out.println("]");
  }

  public void mutate(double[] array) {
    for (int i=0; i<(array.length/2); i++) array[i] = 0;
  }

  public static void main(String[] args) {
    double[] d = {2, 2, 2};
    PassByTest b = new PassByTest();
    System.out.print("Before mutate: ");
    b.printArray(d);
    b.mutate(d);
    System.out.print("After mutate:  ");
    b.printArray(d);
  }
}
```

Output:

```
Before mutate: [ 2.0 2.0 2.0 ]
After mutate:  [ 0.0 2.0 2.0 ]
```

Why is there only a **0** in the first element?!

The resulting value is a **double** but the array indexes are **int**s.

The array itself in **main()** has been mutated. (We called **print()** from the array in **main()**!)

Queen Mary University of London

… and things for you to try out!

Javadocs

- The idea of javadocs is to be able to easily generate a code "maintenance manual".
- Run the **`javadoc`** tool over the source code, to parse the declarations and Doc comments, generating a set of HTML pages.

- General form of Doc comments
  - A Doc comment is made up of two parts: a description, followed by zero or more tags, with a blank line (containing a single asterisk (**`*`**)) between these two sections.
  - Example:

```
/**
 * This is the description part of a Doc comment.
 *
 * @tag Comment for the tag
 */
```

Order and Examples of Doc Tags

- Include tags in the following order:

```
* @author (classes and interfaces only, required)
* @version (classes and interfaces only, required)
*
* @param (methods and constructors only)
* @return (methods only)
* @exception
* @see
* @since
* @serial (or @serialField or @serialData)
* @deprecated
```

JDK Docs and Example

- External documentation can be created with Javadoc comments.
- To generate **javadoc** documentation, type on the command line:

  javadoc -d docs file.java

- View the docs with a web browser:
  - Start with the **index.html** file in the **docs** subdirectory.

```
/**
 * This class counts to a number dictated
 * by the input to the program.
 * Copyright: Copyright (c) 2001
 * @author  Laurissa Tokarchuk
 * @version 1.0
 */
public class CountTo {
  /**
   * This method counts from zero to X,
   * printing the “count” to the screen.
   * @param countTo  int number to count to.
   */
  public void count(int countTo) {
    for (int i = 0; i < countTo; i++)
      System.out.println(“Count = ” + (i+1));
  }
  public static void main(String[] args) {
    new CountTo().count(5);
  }
}
```

… and things for you to try out!