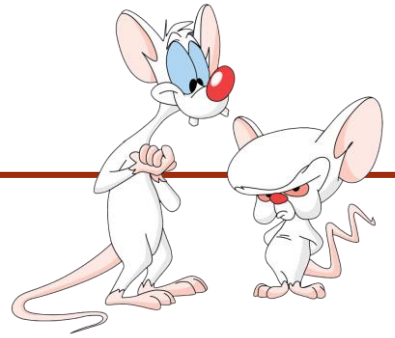
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 2-3: Reference types and Random

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- **Reference types**
- `Random`

REFERENCE TYPES

PRIMITIVE VS REFERENCE TYPES

- Both arrays and Strings are Objects.

- In java, except for the primitive data types (those whose names start with lowercase letters like `int`, `double`, etc.), everything is an `Object`.

- Variables of `Object`s, arrays included, don't store the values of the objects, but a **reference** to the location in memory containing that value. You can think of it as an address which points to where the data is located in memory.

REFERENCE TYPES

```
1  public class Test {
2
3      public static void main(String[] args) {
4          String[] pets = {"cats", "dogs", "ferrets"};
5          System.out.println(pets);
6
7          int[] x = new int[5];
8          System.out.println(x);
9      }
10 }
```


Problems Javadoc Declaration Console

<terminated> Test (11) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Jan. 11, 2020, 3:51:49 p.m.)

```
[Ljava.lang.String;@15db9742
[I@6d06d69c
```

PRIMITIVE VS REFERENCE TYPES – EXAMPLES

```
public static void main(String[] args) {
    int x = 5;
    int y = x;
    x++;
    System.out.println(x + " " + y);
}
```


- **What does the program print?**

```
6 5
```

PRIMITIVE VS REFERENCE TYPES – EXAMPLES

```
public static void main(String[] args) {
    int[] x = {1, 2, 3};
    int[] y = x;
    y[0] = 4;
    System.out.println(x[0] + " " + y[0]);
}
```


- What does the program print?

```
4 4
```

PRIMITIVE VS REFERENCE TYPES – EXAMPLES

```
public static void main(String[] args) {
    int x = 5;
    example(x);
    System.out.println(x);
}
public static void example(int x) {
    x = x*5;
}
```


- What does the program print?

5

PRIMITIVE VS REFERENCE TYPES – EXAMPLES

```
public static void main(String[] args) {
    int[] x = {1,2,3};
    example(x);
    System.out.println(x[0]);
}
public static void example(int[] x) {
    x[0] = 4;
}
```


- **What does the program print?**

```
4
```

ARRAY VS STRING

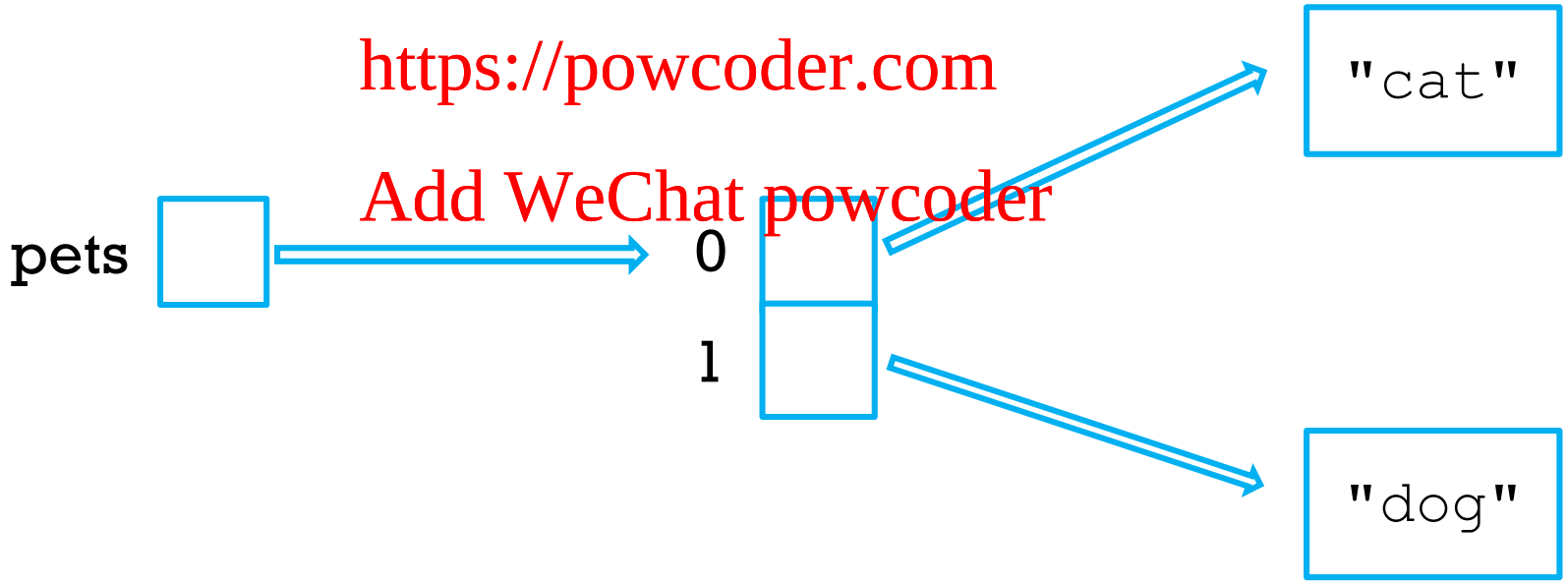
- Both arrays and strings are reference types.

- Variables of array-type and String-type both store the address in memory at which the elements of the object begins.

- Arrays are mutable, Strings are immutable!!
  - Once a String has been created it cannot be changed!
  - The elements of an array can be updated anytime we want.

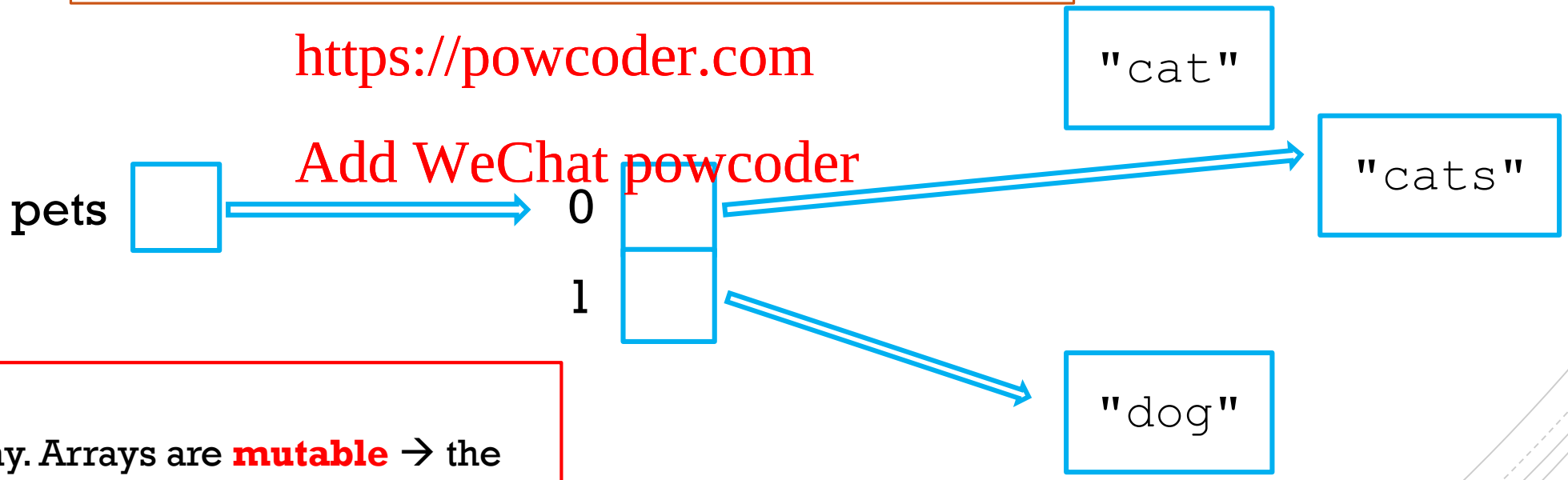
REFERENCE TYPES

```
String[] pets = {"cat", "dog"};
```

REFERENCE TYPES

```
String[] pets = {"cat", "dog"};
pets[0] = pets[0] + "s";
```

To note:

- We changed the array. Arrays are **mutable** → the reference stored in pets did not change!
- We changed the first String. Strings are **immutable** → the reference in pets[0] did change!

ARRAY VS STRING – EXAMPLE 1

```
public static void main(String[] args) {
    int[] x = {1, 2, 3, 4};
    myMethod(x);
    System.out.println(Arrays.toString(x));
}

public static void myMethod(int[] a) {
    for(int i=0; i<a.length; i++) {
        a[i] = i;
    }
}
```


- **What prints?**
  - `[0, 1, 2, 3]`

ARRAY VS STRING – EXAMPLE 1

```
public static void main(String[] args) {
    String s = "word";
    myMethod(s);
    System.out.println(s);
}

public static void myMethod(String t) {
    t = t + "s";
}
```


- **What prints?**
  - `word`

ARRAY VS STRING – EXAMPLE 2

```
char[] letters = {'w', 'o', 'r', 'd'};
for(int i=0; i<letters.length; i++){
   if(letters[i]=='o') {
      letters[i] = 'a';
   }
}
System.out.println(Arrays.toString(letters));
```


- **What prints?**
  - `[w, a, r, d]`

ARRAY VS STRING – EXAMPLE 2

```
String s = "word";
for(int i=0; i<s.length(); i++) {
   if(s.charAt(i)=='o') {
      s.charAt(i) = 'a';
   }
}
System.out.println(s);
```


- **What prints?**
  - `compile time error: unexpected type.`
    `Required: variable. Found: value.`

ARRAY VS STRING – EXAMPLE 2

```
String s = "word";
String t = "";
for(int i=0; i<s.length(); i++) {
   if(s.charAt(i)=='o') {
      t = t + "a";
   } else {
      t = t + s.charAt(i);
   }
}
System.out.println(t);
```

THE NULL KEYWORD

- Any reference type variable can have a **`null`** value.

- `null` indicates the absence of an address.

- We can think of a variable with value `null` as a box with no arrow/pointing nowhere.


```
int[] blank = null;
```

NullPointerException

- If we try to access information through a variable with value `null`, the code will throw a `NullPointerException`.


```
int[] blank = null;
System.out.println(blank.length);
System.out.println(blank[0]);
```

DEFAULT VALUES

- In java, **local variables** (those declared within the body of a method, i.e. all the variables we have seen up to now) are **not** given an initial default value!
  - This is why if we try to use a variable without initializing it, the compiler will throw the following error: "variable ___ might not have been initialized"
- On the other hand, **array elements** (and other kind of variables, tbd) are initialized with default values:
  - int/short/byte/long with 0
  - double/float with 0.0
  - boolean with false
  - char with 0
  - **reference types with `null`.**

EXAMPLES – LOCAL VARIABLES

```
int base;
int area = squareArea(base);
```

```
String day;
System.out.println("Today is " + day);
```

```
int[] grades;
int size = grades.length;
```

Compile-time error! Variable not initialized!

EXAMPLES – ARRAYS' ELEMENTS

```
int[] num = new int[3];
int sum = num[0] + num[1] + num[2];
```

sum has value 0


```
String[] days = new String[7];
System.out.println("Today is " + days[4]);
```

> Today is null

```
String[] days = new String[7];
int numLettersMonday = days[0].length();
```

NullPointerException

RANDOM

THE RANDOM CLASS

- Up to now you probably learned how to use `Math.random()` to get random numbers between a minimum value and a maximum value.

- We can also use the `Random` class to generate random numbers.

- The `Random` class allows us to ***seed*** the random numbers such that we will see ***the same*** sequence of 'random' numbers each time.
  - Why is it useful?
    Easier to debug code that is not working.
    Comparing outputs from different codes (for instance your assignments)

HOW TO USE RANDOM

First import the `Random` **class: add** `import java.util.Random;`

Then you can create a random number generator using the following statement:


```
int seed = 123;
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

HOW TO USE RANDOM

First import the `Random` **class: add** `import java.util.Random;`

Then you can create a random number generator using the following statement:


```
int seed = 123;
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

Declaration of two variables of type `Random`**.**

HOW TO USE RANDOM

First import the `Random` **class: add** `import java.util.Random;`

Then you can create a random number generator using the following statement:


```
int seed = 123;
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

Declaration of two variables of type `Random`**.**

Creation of a `Random` **object. Note the** `new` **keyword! Random is a reference type.**

METHODS IN RANDOM

boolean	**nextBoolean()** Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
void	**nextBytes**(byte[] bytes) Generates random bytes and places them into a user-supplied byte array.
double	**nextDouble()** Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number ge
float	**nextFloat()** Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number ger
double	**nextGaussian()** Returns the next pseudorandom, Gaussian (normally) distributed double value with mean 0.0 and standard deviation
int	**nextInt()** Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
int	**nextInt**(int n) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), dra
long	**nextLong()** Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.
void	**setSeed**(long seed) Sets the seed of this random number generator using a single long seed.

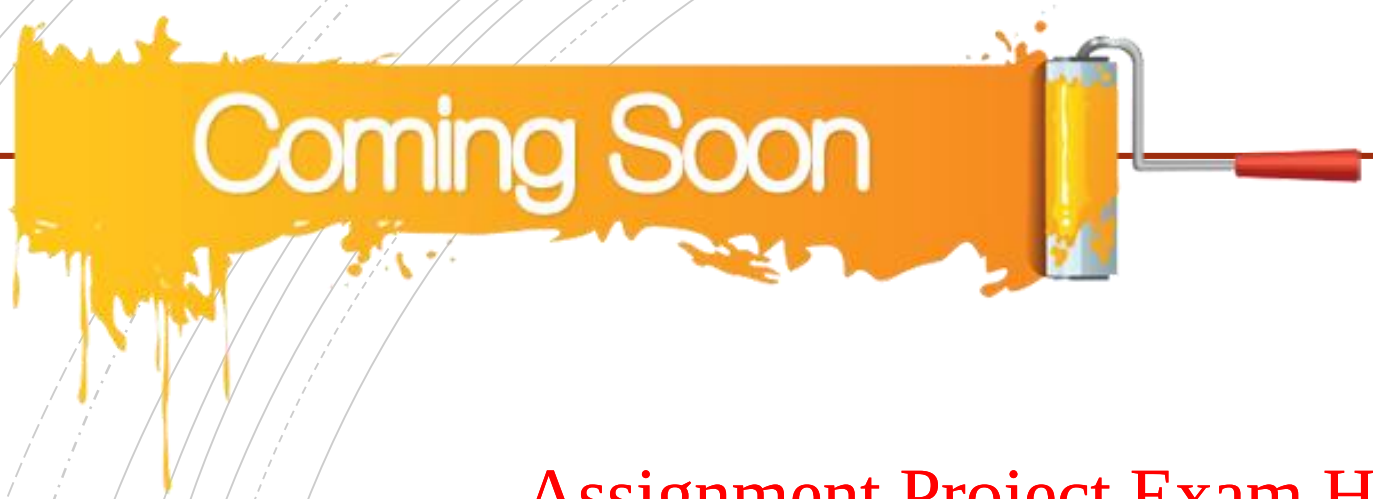
https://docs.oracle.com/javase/7/docs/api/java/util/Random.html

DICE ROLL

Here's an example of using `Random` to simulate a dice roll


```
Random randomGenerator = new Random();
int diceRoll = randomGenerator.nextInt(6) + 1;

System.out.println("The dice rolled " + diceRoll);
```



In the next video we will be talking about errors and exceptions, as well as try/catch blocks.