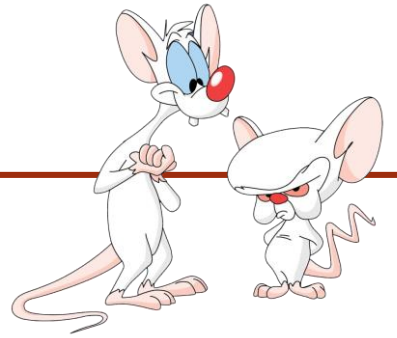
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 2-3: Reference

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- **Reference types**
- `Random`

RE ES

PRIMITIVE VS REFERENCE TYPES

- Both arrays and Strings are Objects.

- In java, except for t (those whose names start with lowercase , 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

PRIMITIVE VS REFERENCE TYPES – EXAMPLES

```
public static void main(String[] args) {
    int x = 5;
    int y =
    x++;
    System.out.println(          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
    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.
}
public static void exa          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.
}
public static void exa          ] x) {
    x[0] = 4;
}
```


- **What does the program print?**

```
4
```

ARRAY VS STRING

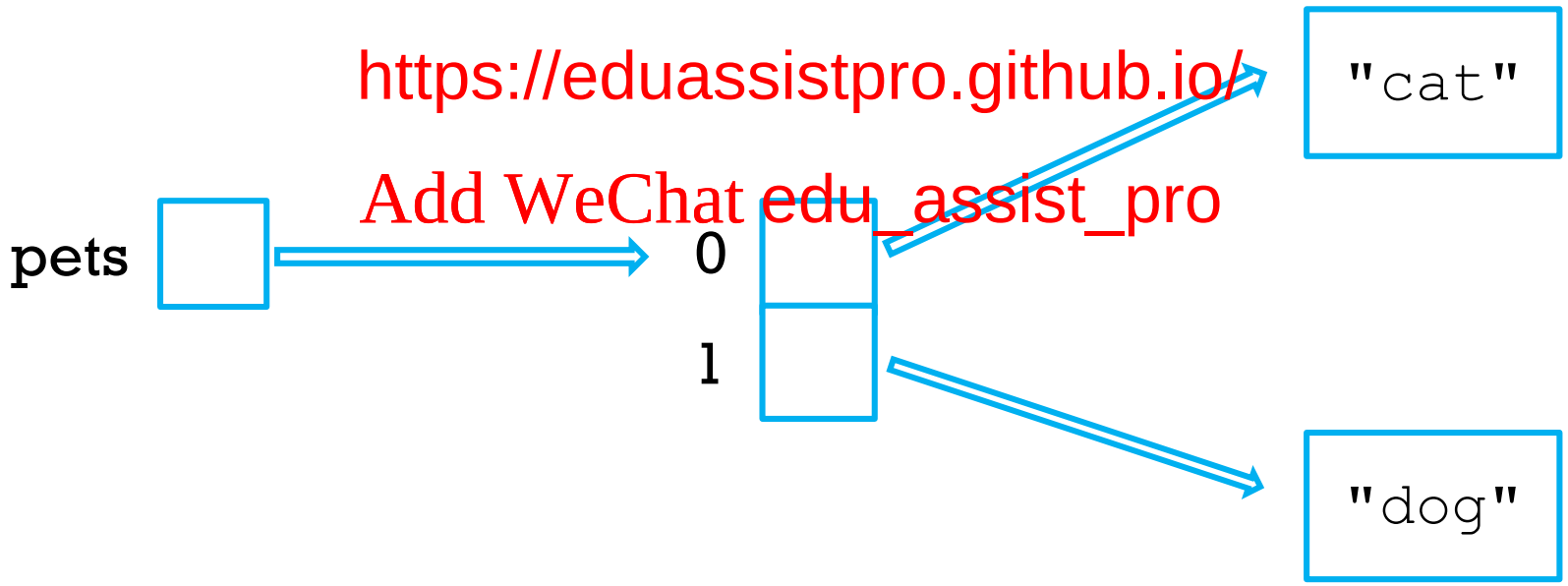
- Both arrays and strings are reference types.

- Variables of array-t store the address in memory at which ect 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.

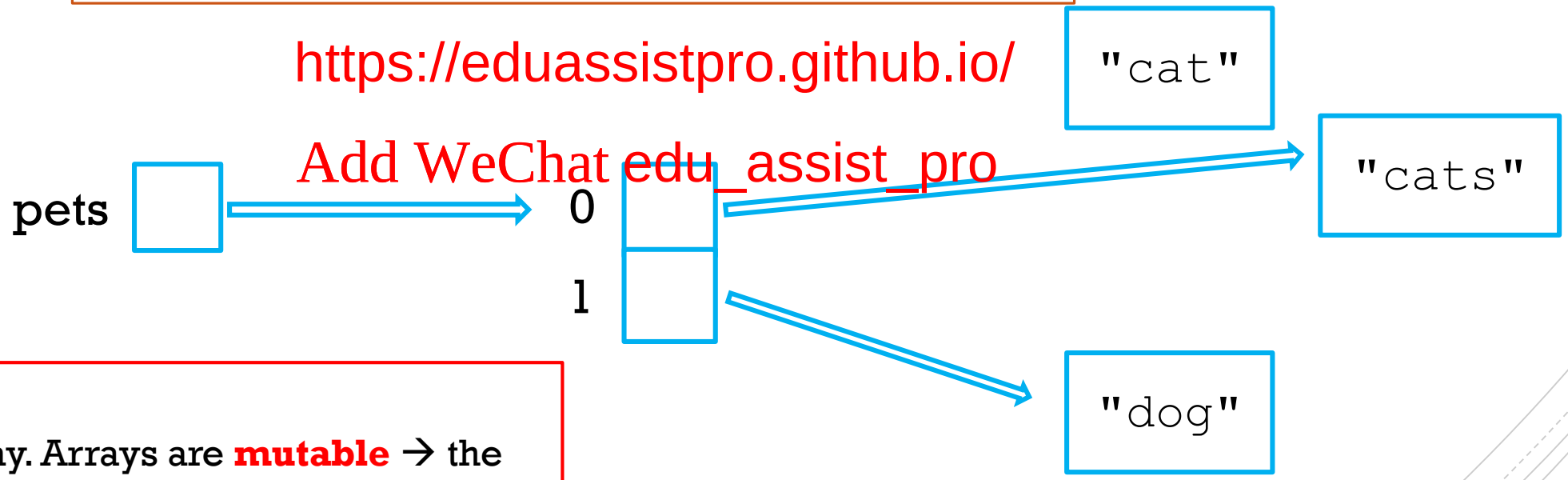
REFERENCE TYPES

```
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 myMe                    a) {
    for(int i=0; i<a.len                   {
        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.
}

public static void myM          ing 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
      letters[
   }
}
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.cha
      s.cha
   }
}
System.out.println(s);
```


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

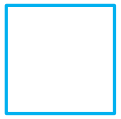
ARRAY VS STRING – EXAMPLE 2

```
String s = "word";
String t = "";
for(int i=0; i<s.length(); i++) {
   if(s.char
      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 a
- We can think of a variable with val a box with no arrow/pointing nowhere.


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

blank 

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 "variable may not have been initialized"
- On the other hand, **array elements** (and other 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 "
```

```
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 " + d
```

> Today is null

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

NullPointerException

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 `R` random numbers.

- The `Random` class allows us to ***seed*** t umbers 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

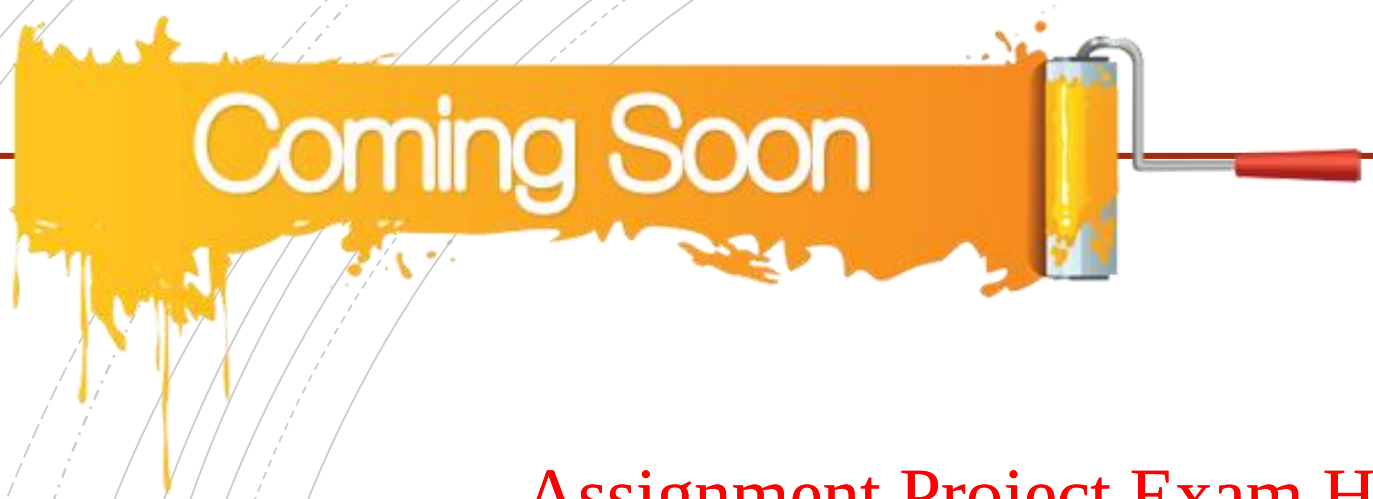
https://docs.oracle.com/javase/7/docs/api/java/util/Random.html

DICE ROLL

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


```
Random randomGenerator = n                ();
int diceRoll = randomGener                Int(6) + 1;

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



In the next g about errors and excep ch blocks.