COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 3-3: OOD 2 C is

Giulia Alberini, Fall 2020

FROM LAST VIDEO

- Packages
- Fields
- Modifiers

MODIFIERS REVIEW – OUTER CLASS

Dog.java

```
package animals;

public class Dog {
    ⋮
}
```

Farm.java

```
package buildings;
import animals.Dog;

        class Farm {
                ;
    ⋮
}
```


Does the compiler allow this?

- Yes

MODIFIERS REVIEW – OUTER CLASS

Dog.java

```
package animals;

class Dog {
   ⋮
}
```

Farm.java

```
package buildings;
import animals.Dog;

        class Farm {
                ;
    ⋮
}
```


Does the compiler allow this?

- **No, the class `Dog` is visible only within its package!**

MODIFIERS REVIEW – FIELDS

Dog.java

```
package animals;

public class Dog {
   public String na
   ⋮
}
```

Farm.java

```
package buildings;
import animals.Dog;
                class Farm {
                    void f() {
                        = new Dog();
                    d.name = "Jessie";
    }
}
```


Does the compiler allow this?

- Yes (but remember, as a general rule fields should be declared private)

MODIFIERS REVIEW – FIELDS

Dog.java

```
package animals;

public class Dog {
   String name;
   ⋮
}
```

Beagle.java

```
package animals;

public class Beagle {
   Dog d;

   void f() {
      d = new Dog();
      d.name = "Buddy";
   }
}
```


Does the compiler allow this?

- **Yes, the field** `name` **is visible within the package** `animals`**.**

MODIFIERS REVIEW – FIELDS

Dog.java

```
package animals;

public class Dog {
   private String n
   ⋮
}
```

Beagle.java

```
package animals;

public class Beagle {
          d;
                  void f() {
          = new Dog();
                name = "Buddy";
    }
}
```


Does the compiler allow this?

- **No,** `name` **is visible only within the class** `Dog`**.**

WARM-UP

- We would like to define a new data type that represents a hospital Patient.
- In this class we'd like to have the following fields:
  - A field to store the na
  - A field to store the age of the patient
  - A field to store the body temperatur ient since the day they were admitted to the hospital
  - A field to store the body temperature considered to be a possible fever.

INITIAL VALUES

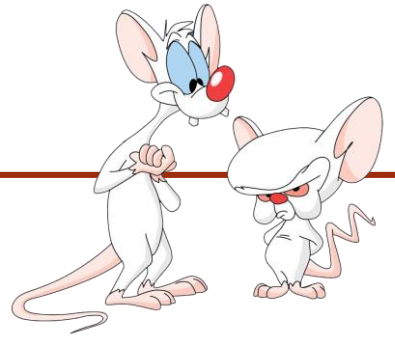
One thing we noticed is that the initial values of the attributes are all set by default (based on their type) when an object is created.

For example


```
Patient x = new Patient();
System.out.println(x.name);          // prints null
System.out.println(x.age); // prints 0
System.out.println(x.temps); // prints null
```


If we want to specify the initial values to be something other than this, we should add a method to our class called constructor.

WHAT ARE WE GOING TO DO IN THIS VIDEO?



OOD2

- **Constructors**
- **Keyword** `this`

C RS

STEP 2

```
public class ClassName {
   // some data declared here
   <modifier> <type> <variable_name>;

   public ClassName(
      //constructor
   }

   // declare other methods
}
```


Data

Method to create an object

Other methods

File name: **ClassName**.java

CREATING OBJECTS

- Remember how we created objects:


```
Random g
Random gen = new Ra
```


- We now need to write the code to create our object.

CONSTRUCTORS

A method that is executed when a new object of that class is created. The syntax for constructors is similar to that of other methods, except:

- The name of the constructor must be the same as the name of the class.
- It has no return type (not even void!)
- It is non-static

CONSTRUCTORS

If you don't write a constructor, the default constructor for a class looks like:


```
pub
}
```


If you write your own constructor, you no longer have access to the default constructor.

TRY IT!

1. Let's write a constructor for the Patient class that takes no input and prints out "Creating a new patient".

2. Add now another constructor to t his constructor should take the name of the patie ir age as input and assigns them to the corresponding attributes attributes.

THE `this` KEYWORD

- **`this`** is a keyword that refers to the object on which the method have been called. In the case of a constructor, **`this`** refers to the current object being created.
- It allows Java to disting nd local variables that have the same name.
- You can use **`this`** the same way you use the name of any other object. For example, you can pass **`this`** as an argument to other methods.
- NOTE: You do not declare **`this`**, and you can't make an assignment to it.

EXAMPLE


```
public class Patient{
    privat
    privat

    public Patient(Str          int age) {
        this.name   = name;
        this.age    = age;
    }
}
```

Attributes!

Local variables!

```
public class Patient{
    private String name;
    private int age;

    public Patient(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void printName() {
        System.out.println(this.name);
    }

    public static void main(String[] args) {
        Patient x = new Patient("John",43);
        x.printName();
    }
}
```


this refers to the object currently being created

this refers to the object on which the method was called.

THE THIS KEYWORD AND STATIC

It ***never*** makes sense to use the `this` keyword in a `static` method.

A `static` method (o to the entire class. The `this` keyword refers to an instance .

OMITTING THIS

You can leave out `this` if there is no naming conflict:


```
public Patient(String name, int age) {
    this.name = na
    this.age = age
}
```



```
public Patient(String patientName, int patientAge) {
    name = patientName;
    age = patientAge;
}
```

OVERLOADING

- In java, we can ***overload*** methods. This happens when we write two methods in the same class, with the same name, but different parameters.
- You cannot overload a the return type/modifiers.


```
public void someMethod(int x)
public void someMethod(int y) { ...
public void someMethod(String s) {...
```

```
public void someMethod(int x) { ...
public int someMethod(int y) {...
```

WHY OVERLOADING?

```
System.out.println("Welcome!");
System.out.println(123);
Syste
System.out.println
```


- **The `println` method is overloaded! We can call the method and pass values with different types as input, and it always compiles.**

WHY OVERLOADING?

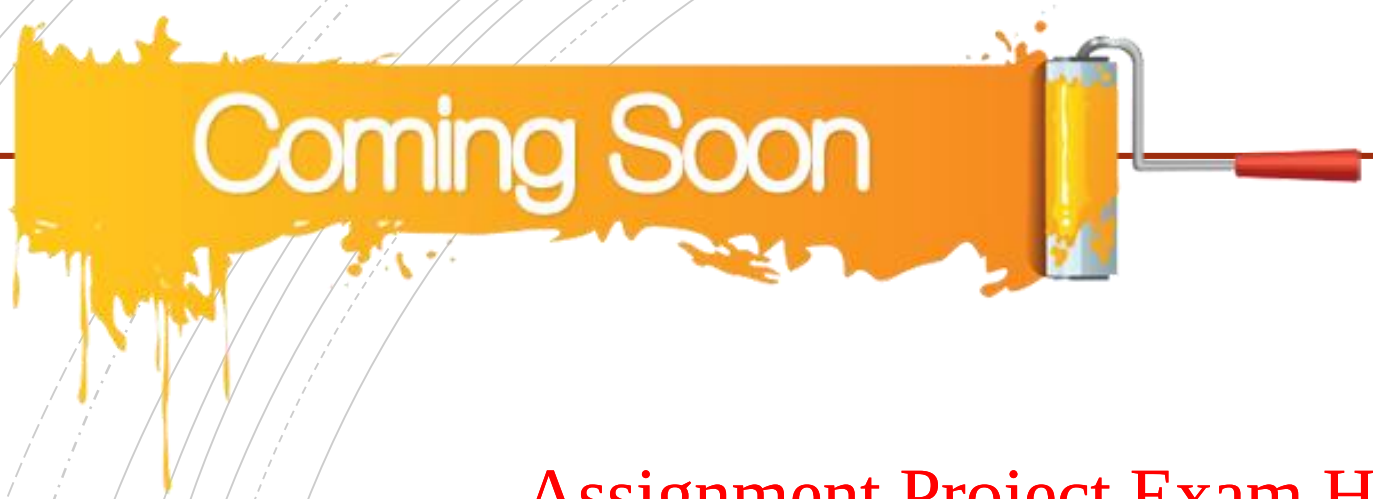

```
Random gen1 = new Random();
Rando                    (42);
```


- We cannot choose the name of the constructor. But we might want a constructor to sometimes take inputs and others not. In this case, we need to overload it.

TRY IT!

- In the `Patient` class, overload the constructor by adding another one that takes as input the name, age, and temperatures of the new patient.

Does it make a difference if we co ents of the input array, or just the address directly?



In the next

- Other m
- Mutable vs immutable

- Final variables