CODE DSM: JAVA WEEK 07 OBJECT ORIENTED PROGRAMMING

DMACC Fall 2019

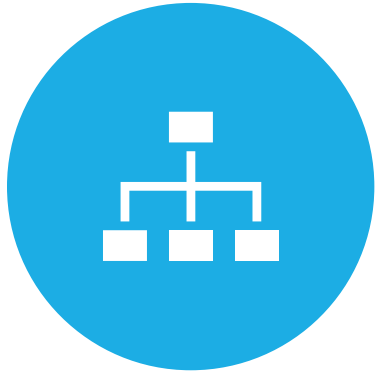
Instructor: Greg Hazen

AGENDA REVIEW

WEEK 6 QUIZ

WEEK 6 ASSIGNMENT

AGENDA



OBJECT ORIENTED PROGRAMMING

OBJECTS

ENCAPSULATION

REVIEW WEEK 6 QUIZ AND ASSIGNMENT

Switch to IntelliJ!

ANOTHER WAY TO CLOSE FILES/RESOURCES

```
Scanner menuScanner = new Scanner(file);
try {
    while (menuScanner.hasNextLine()) {
        String line = menuScanner.nextLine();
        String[] split = line.split(",");
        menu.put(split[0], Double.parseDouble(split[1]));
    }
} finally {
    menuScanner.close();
}
```

ANOTHER WAY TO CLOSE FILES/RESOURCES

```
try (Scanner menuScanner = new Scanner(file)) {
    while (menuScanner.hasNextLine()) {
        String line = menuScanner.nextLine();
        String[] split = line.split(",");
        menu.put(split[0], Double.parseDouble(split[1]));
    }
}
```

PROCEDURAL PROGRAMMING

So far we've been doing procedural programming

All written in one file

Instructions run top down

- Enter main
- Do stuff
- Exit

PROCEDURAL PROGRAMMING

In week 6 we made a restaurant all in main

1. Load menu
2. Ask customer what to do
3. Print the menu or total
4. Exit

OBJECT ORIENTED PROGRAMMING

Switching to an Object Oriented mindset

- Create a name that describes a concept
- List any state that it needs to keep track of
- List behaviors that other can use
- List behaviors that others don't need to know

Let's see an example…

OBJECT ORIENTED PROGRAMMING

Switching to an Object Oriented mindset

- Create a name that describes a concept
  - Restaurant Menu
- List any state that it needs to keep track of
  - Items, Prices
- List behaviors that other can use
  - Add to Menu, Order from Menu
- List behaviors that others don't need to know
  - Load from File, Save to File, Handle if File doesn't Exist…

OBJECT ORIENTED PROGRAMMING

Consider

- Is the concept name general enough to reuse?
- What data should be accessible and what should be hidden?

OBJECTS

Used to model

- The state of a system using data
- The behaviors and interactions between parts of a system

OBJECTS

We define a ***class*** that describes the **state and behavior**

When we ***new*** it up, we create an **object or instance** of that ***class***

For example, ***class*** Scanner

- Defines the **Behavior** of nextInt(), nextLine()…
- Maintains the **state** of reading from the console

new Scanner() creates an **object or instance** of that class

EXAMPLE

Let's refactor week 6 assignment

Instead of all logic being in *main*, let's

1. Create a *Menu class*
2. Create an *instance* of the *Menu class* in *main*
3. Use the *instance* to simplify our *main*

OBJECTS

```
public class Menu { // Class names start with Uppercase
    File file;
    HashMap<String, Double> menu;
    void loadMenuFromFile() { … }
    void saveMenuToFile() { … }
    void showMenu() { … }
    void addToMenu(String item, double price) { … }
}
```

State

Behavior

INSTANCE

```
public class Menu {
    File file;
```

If we have a menu for breakfast and a menu for dinner, each Menu object is a unique **instance** of the Menu class.

```
    Menu breakfastMenu = new Menu();
    Menu dinnerMenu = new Menu();
```

INSTANCE VARIABLES

```
public class Menu {
    File file;
```

file is an **instance variable** or **member variable,** so *breakfastMenu.file; and dinnerMenu.file;* return to completely separate values

METHODS

```
public class Menu {
    void showMenu() { … }
```

Methods in the class describe the behavior

Methods can access the **instance variables**, so

breakfastMenu.showMenu() and dinnerMenu.showMenu()

- Use the same behavior, but
- Use completely separate menu values

LET'S REVIEW THE WORDS WE'VE LEARNED…

Class describes the state and behavior of a concept

Objects are an **instance** of a class

- We can have multiple **instances** of a class

Variable stores a value or instance of an object

Instance Variable stores a value specific to an instance

- breakfastMenu.file and dinnerMenu.file return two different files

CONSTRUCTOR

Called when you create a *new* instance of a class

```
new Menu(); // calls the constructor
public class Menu {
    Menu() { // Constructor
        // Do something when instance is created
    }
}
```

CONSTRUCTOR

Has the same name as the class, including the uppercase letter

Can take any number of parameters

CONSTRUCTOR

Used to initialize values in that instance

If you don't initialize instance variables, they initialize automatically to their default value

Type	Default Value
Any Object (String, Car, Bank…)	null
Primitive byte, short, int, long, double, float, char	zero
Primitive boolean	false

DEFAULT CONSTRUCTOR

If you don't create a constructor, the class has a default constructor with no parameters that does nothing extra

```
public class Menu {
    Menu() {} // Created by default if you don't create one
}
```

CONSTRUCTOR

If you do provide a custom constructor, then no default constructor is created

```
public class Menu {
    Menu() { // Can add zero or more parameters
        // Do something when instance is created
    }
}
```

EXERCISE

Create a constructor in your *Menu class*

- Take in no parameters
- Initialize *file* and *menu* in the constructor

EXERCISE

Update the restaurant application to ask which meal they want (e.g. ‘breakfast’ or ‘dinner’)

- Add *meal* as a parameter to the *Menu* constructor
- In the constructor, change *file* to use the *meal* parameter in the file name

CONSTRUCTOR CONT…

If your constructor or method takes a parameter with the same name as an instance variable, use '*this*' to access the instance variable

```
String meal;

Menu(String meal) {
        this.meal = meal;
}
```

CONSTRUCTOR CONT…

Constructors can call eachother

```
public class Counter {
        private int count;
        public Counter() {
                this(0);
        }
        public Counter(int initialCount) {
                this.count = initialCount;
        }
}
```

PROBLEM WITH EXPOSING THE DETAILS

What if another object emptied our menu?

What if we changed to using a database instead?

VARIABLE ENCAPSULATION

Publicly providing an interface that other objects can use while hiding the implementation details

Other objects don't need to know if the menu is saved to a file or a database

If we switch to using a database, other objects using the Menu class shouldn't have to change too

VARIABLE ENCAPSULATION

Basic rule of thumb:

Don’t expose the instance variables that store the state of your object

Instead, expose methods to interact with the instance variables

This puts your object in control and more predictable

VARIABLE ENCAPSULATION

Hide your instance variables from other objects using access modifiers

- public: any object can modify it
- private: can only modify it from inside the instance
- No Modifier: classes in the same folder/package can modify it

VARIABLE ENCAPSULATION

```
public class Menu {
        private File file;
        private HashMap<String, Double> menu;
        Menu() { … }
        void loadMenuFromFile() { … }
        void saveMenuToFile() { … }
        void showMenu() { … }
        void addToMenu(String item, double price) { … }
}
```

VARIABLE ENCAPSULATION

Typical to create

- ‘get’ methods to read state
- ‘set’ methods to write state

```
private int count = 0;

int getCount() { return count; }

void setCount(int count) { this.count = count; }
```

METHOD ENCAPSULATION

You should also think about how you expose methods

Even if the implementation of your method changes, objects using it shouldn't have to change

METHOD ENCAPSULATION

```
public class Menu {
    private File file;
    private HashMap<String, Double> menu;
    Menu(String meal) { … }
    void loadMenuFromFile() { … }
    void saveMenuToFile() { … }
    void showMenu() { … }
    void addToMenu(String item, double price) { … }
}
```

We wouldn't want other objects to have to pass in a File/Path

Naming shouldn't be implementation specific

METHOD ENCAPSULATION

You can also hide methods that others shouldn't see

If we have a method for formatting the menu, we may want to hide it so that developers use our higher level method instead

```
public class Menu {
    void showMenu() { … }
    void formatMenu() { … }
}
```

METHOD ENCAPSULATION

Hide methods using access modifiers

- public: any object can use it
- private: can only be used from inside the instance
- No Modifier: classes in the same folder/package can use it

```
public class Menu {
        public void showMenu() { … }
        private void formatMenu() { … }
}
```

METHOD ENCAPSULATION

Constructors can also have access modifiers

More complicated use cases will be covered later in the semester

For now just use public

EXERCISE

Update the instance variables, constructor and methods in Menu to be private or public

STATIC KEYWORD

Static is only used for interactions that are ‘constant’ or otherwise unchanging

If you always use a database called MY_DB, you can create a ***constant***

- *public static final String DATABASE_NAME = “MY_DB”;*

System.in and System.out

STATIC KEYWORD

If you have a method that doesn't require an instance of a class to store state and many objects need to refer to it, you can make the method static

- *String.format("$%.2f", price);*

NULL REFERENCE

All objects are initialized to null by default

If you call a method on a null object, you get a NullPointerException

```
Car myCar; // Same as myCar = null;
myCar.start(); // Throws NullPointerException
```

NULL REFERENCE

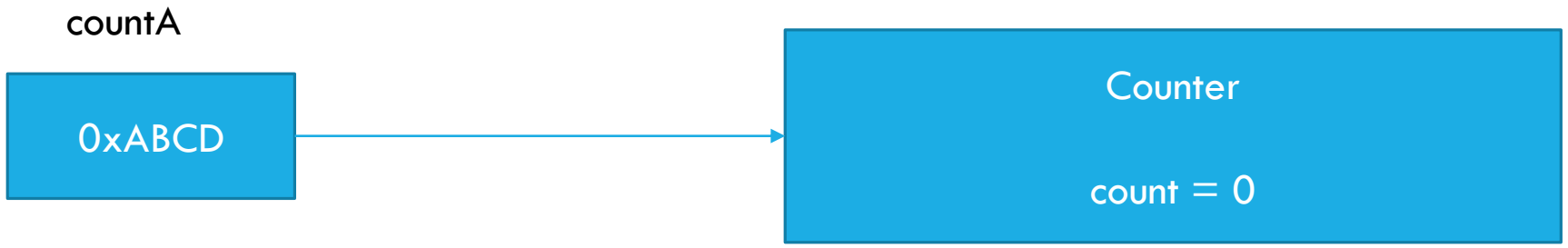
You can use null to know if the value doesn't exist

```
Car car1 = new Car();
Car car2 = null;
If(car2 != null) {
    System.out.println("Dude, you have two cars!");
}
```

STORING A REFERENCE TO AN OBJECT

A variable actually stores a pointer (or the memory address of) an object

Counter countA = new Counter();

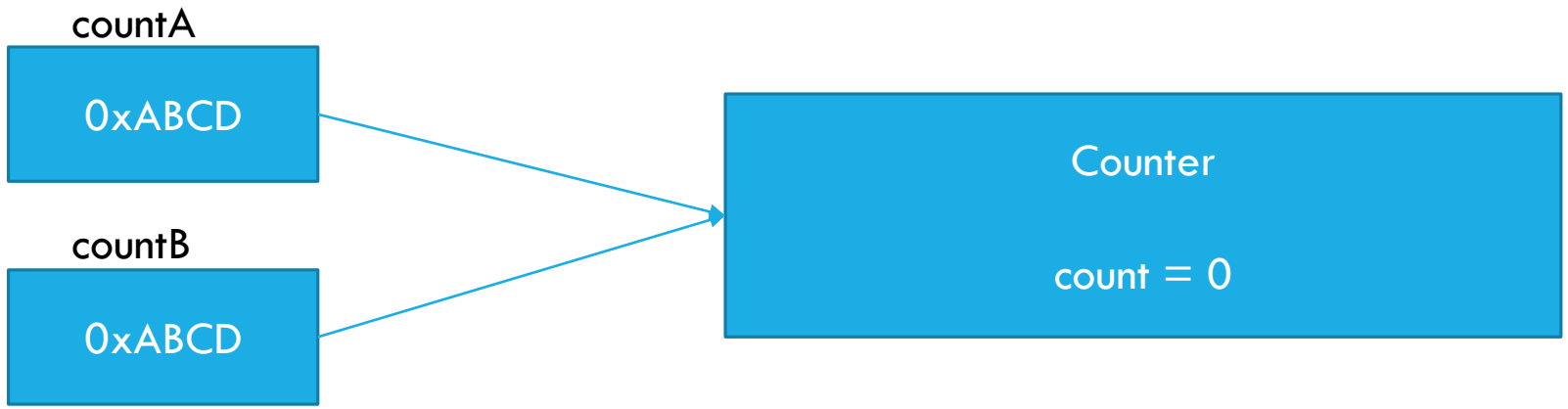


SHARING REFERENCES TO AN OBJECT

Two variables can point to the same object

Counter countA = new Counter();

Counter countB = countA;

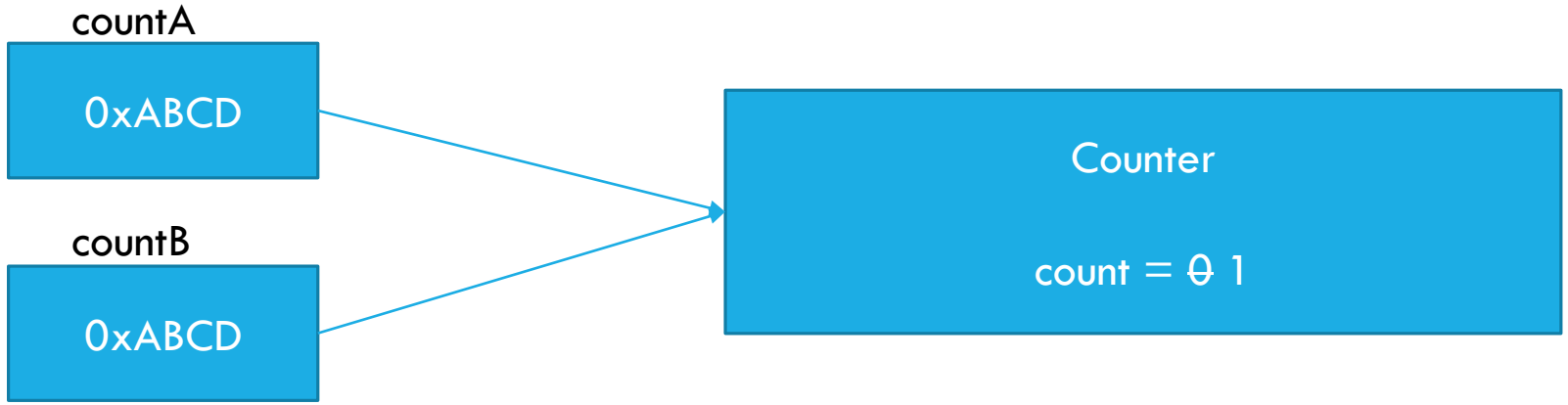


SHARING REFERENCES TO AN OBJECT

When you update through one reference, both are updated because they point to the same reference

countA.increment();

countB.getCount(); // returns 1



ASSIGNMENT

Review assignment for any clarifications

Clone it together

MIDTERM EXAM

Covering week 1 (What is Programming) through week 6 (Handling Exceptions and Files)

Take home, unlimited time, due after Thanksgiving break

Available for you to begin!

QUIZ

Do NOT use IntelliJ for this week's quiz

SEE YOU NOVEMBER 22ND!

Don’t forget to ask question early in the week