Any immediate questions before our session?

Please raise your hand if you have ☺️

Image source: https://www.kindpng.com/imgv/oRomoh_raised-hands-png-transparent-hand-emoji-png-download/

Classes and Objects

Outline

- Formally defining a class and an object
- Access modifiers
  - Class-level
  - Property-/method-level
- Encapsulation
- Constructors

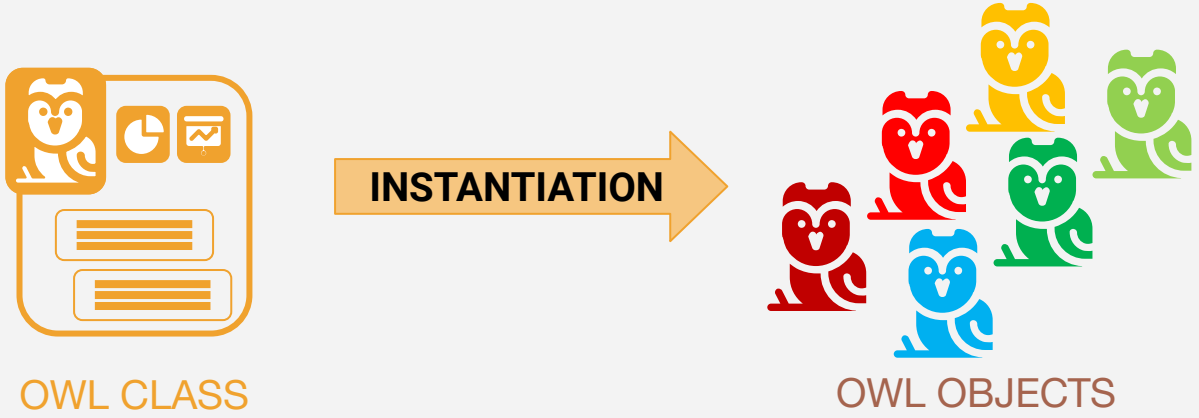
What is a **Class**?

- A class is a structure that defines the data and the methods to work on that data
- It serves as a blueprint for an object
- It has attributes (properties) and methods (behaviors)

What is an **Object**?

- An object or instance is an executable copy of a class
- There can be any number of objects of a given class, in memory, at any one time
- Objects have a copy of the attributes and methods of the class it was instantiated from

Classes and Objects




```
Owl owl1 = new Owl("red");
Owl owl2 = new Owl("blue");
```

Creates a new instance

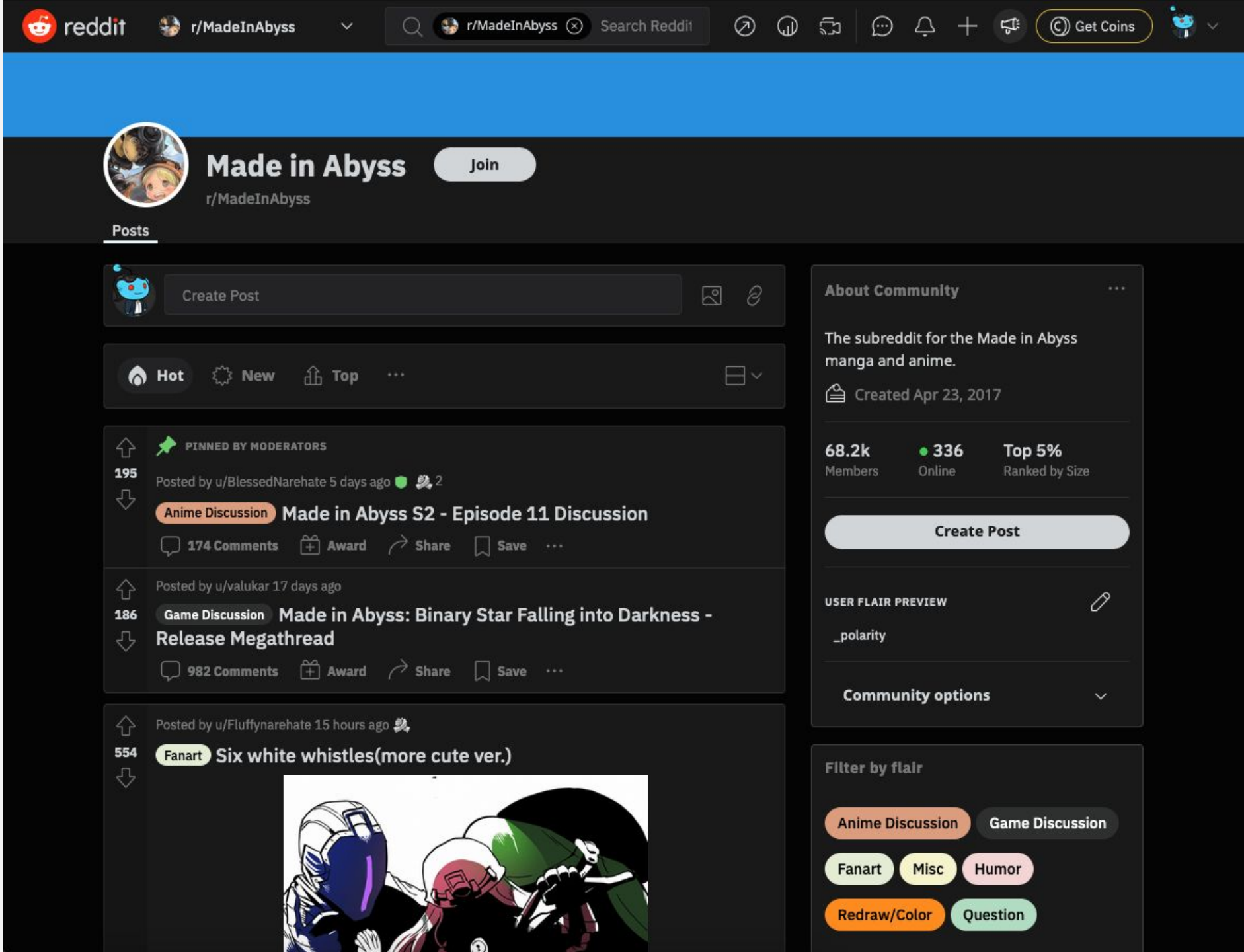
It was mentioned before that it’s easier to model scenarios in OOP…

For example, let’s take a look at the following environment…

What objects exist within this environment?

- Subreddit
- Post
- Comment
- User
- Awards

Most of the above are entities of the environment, but there are also UI objects, like buttons

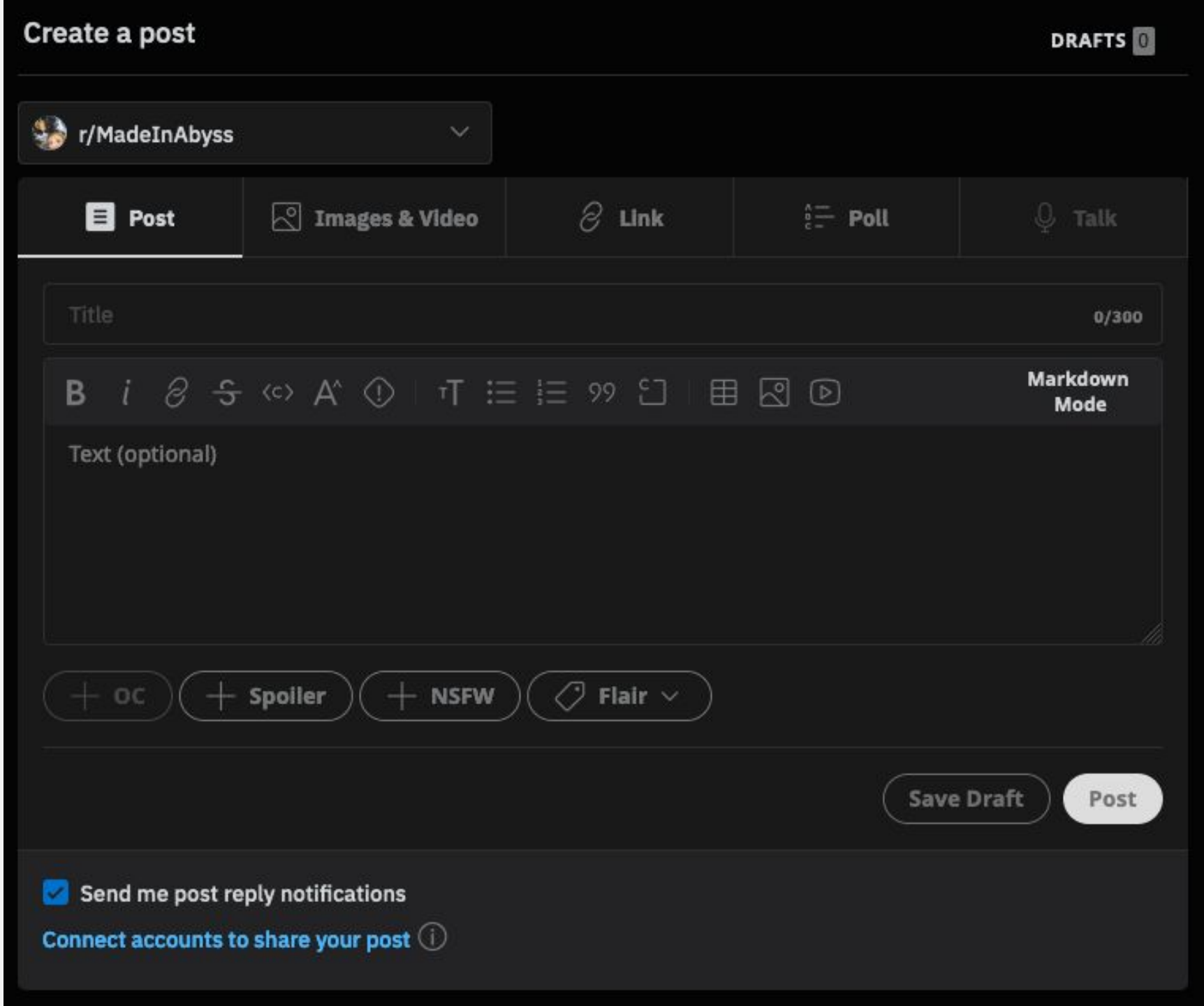


Let's focus on a Post

What characteristics does a post have?

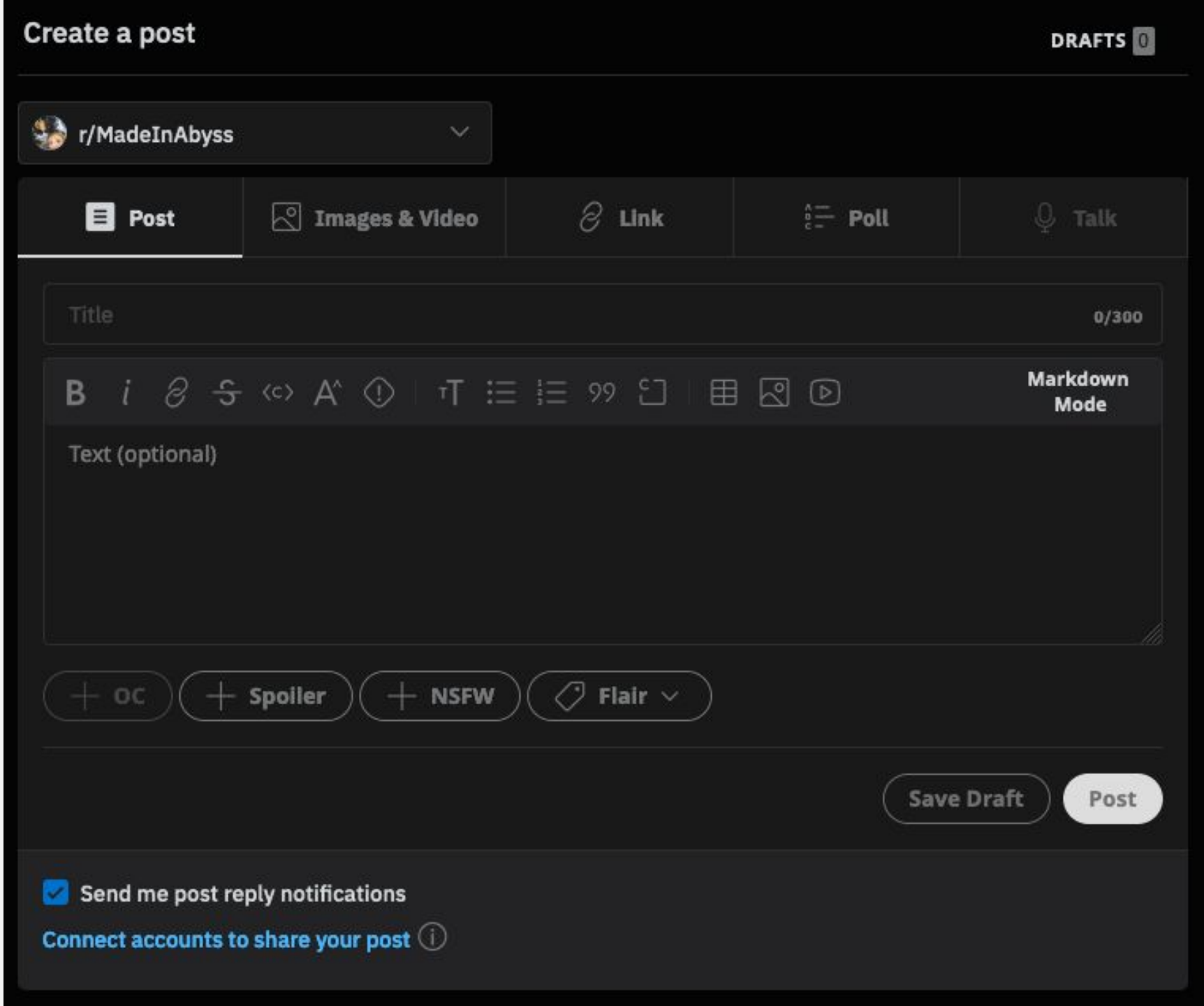
- Title
- Text/caption
- Image/Video
- Spoiler tag
- Post id
- Time stamp
- Etc.

Some characteristics are required, while others are optional (can be null).



Hypothetically, what action/methods might a post have?

- getTitle() : String
  - Gain access to the title for displaying on screen
- isSpoiler() : Boolean
  - If true, blur post; otherwise, no blur
- setText(String text) : void
  - Useful for editing the post's text



We've already been creating classes, but let's first formalize some of the ideas we've been working with

Declaring a Class

- When declaring a class in Java, these components are written in order
  1. The class access modifier
  2. The keyword abstract or final, if declaring an abstract or final class
  3. The keyword *class* followed by the class name, with the initial letter capitalized by convention
  4. The keyword extends followed by the parent class name, if any
  5. The keyword implements followed by a comma-separated list of interfaces implemented by the class, if any
  6. The class body, surrounded by braces { }

Declaring a Class

For now, we'll focus on #s 1, 3, and 6
#s 2, 4, and 5 will be discussed in time

- When declaring a class in Java, these components are written in order
  1. The class access modifier
  2. The keyword **abstract** or **final**, if declaring an abstract or final class
  3. The keyword *class* followed by the class name, with the initial letter capitalized by convention
  4. The keyword **extends** followed by the parent class name, if any
  5. The keyword **implements** followed by a comma-separated list of interfaces implemented by the class, if any
  6. The class body, surrounded by braces { }

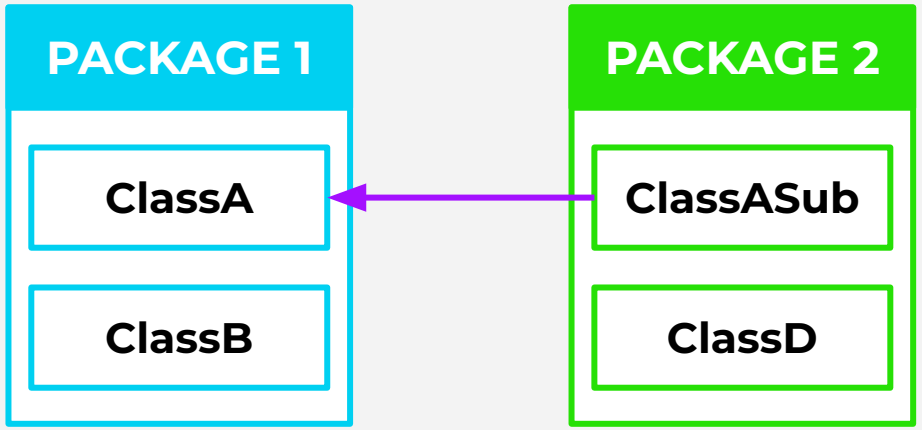
Class Access Modifiers

- There are three you'll need to keep in mind
  - Public
  - Private
  - Default (*err… no access modifier*)

Class Access Modifiers – Public

- The public class access modifier allows the class to be accessed by any other class
- If ClassA is public, all the other classes can access it. This includes classes from other packages (folders)

IF **ClassA** IS **PUBLIC...**

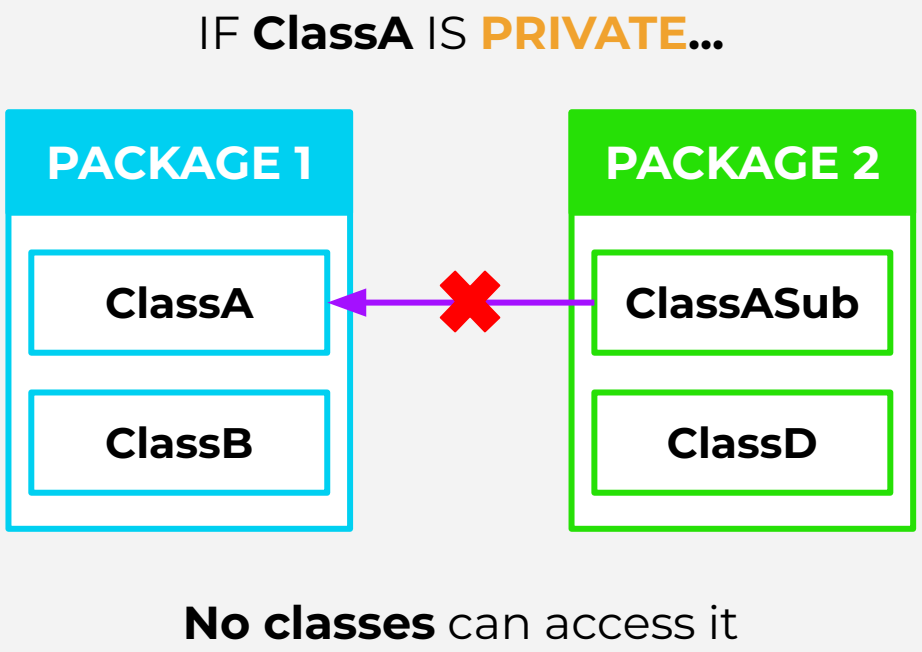


All classes can access it

On Packages, think of these as directories that help organize your code
java.util.Scanner Scanner is a class that's part of the java.util package

Class Access Modifiers – Private

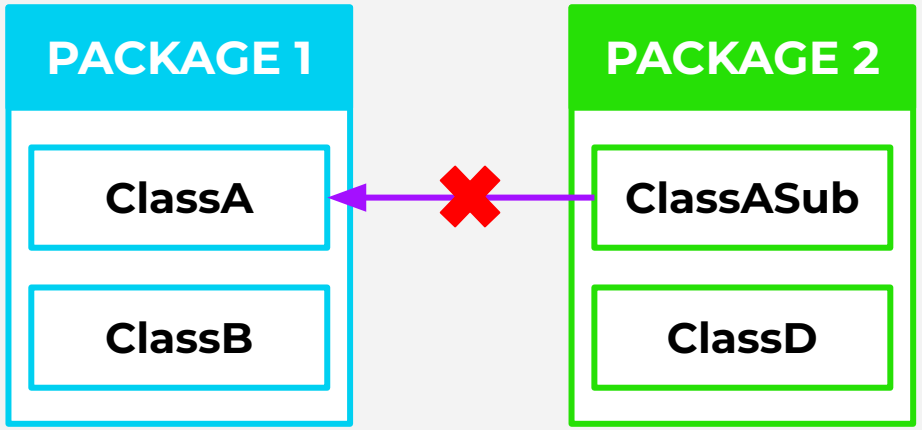
- The private class access modifier doesn't allow the class to be accessed by any other class



Class Access Modifiers – Default

- The default class access modifier is used whenever no access modifier is specified
- This modifier is also called package-private
- Not all OOP languages support default

IF **ClassA** HAS **NO MODIFIER...**



All classes in **Package 2** can't access it.
This includes the subclass

Class Access Modifiers

- There are three you'll need to keep in mind
  - Public
    - A class declared with the public modifier is visible to all classes of any package
  - Private
    - A class can't be declared private, unless it's an inner class. If so, only its outer class can access it
  - Default (*err… no access modifier*)
    - A class with no modifier is only visible within its own package. Even subclasses from another package can't access it

While we're talking about access modifiers, note that all attributes and methods have access modifiers

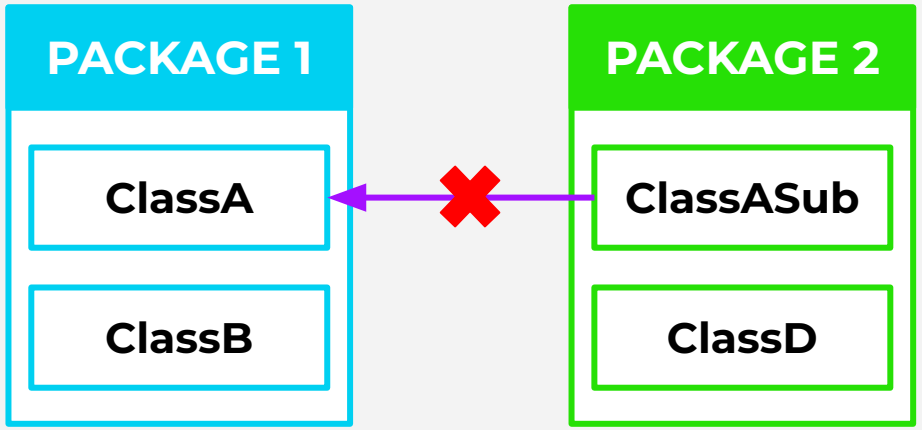
Attribute and Method Access Modifiers

- There are four you'll need to keep in mind
  - Public
  - Private
  - Protected
  - Default (*no access modifier*)

Class Access Modifiers – Private

- Methods or variables declared as private are only accessible within the class they are declared in
- Even classes of the same package will not be able to access these members

A **PRIVATE** METHOD OR PROPERTY IN **ClassA...**



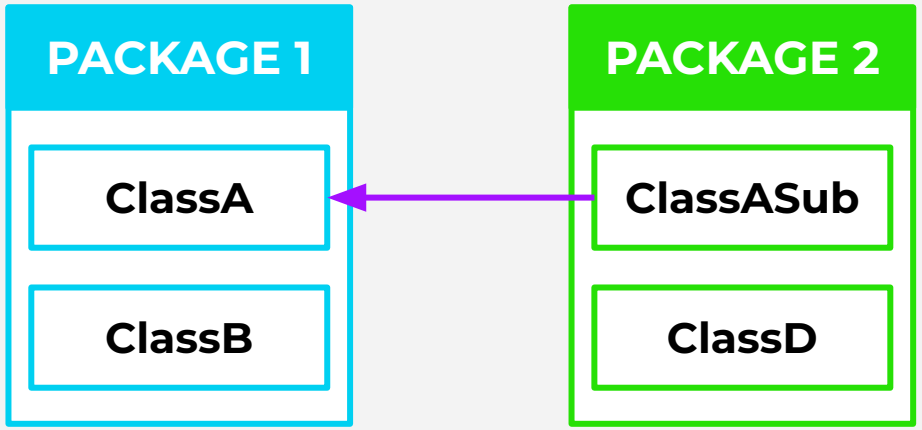
Can't be accessed by **any class** other than **ClassA** itself

Class Access Modifiers – Protected

- Methods or variables declared as protected are accessible by subclasses within any package
- Classes can't be declared protected. This access modifier is generally used in a parent-child relationship

We'll discuss more of this when we enter inheritance ☺️

A **PROTECTED** METHOD OR PROPERTY IN **ClassA...**

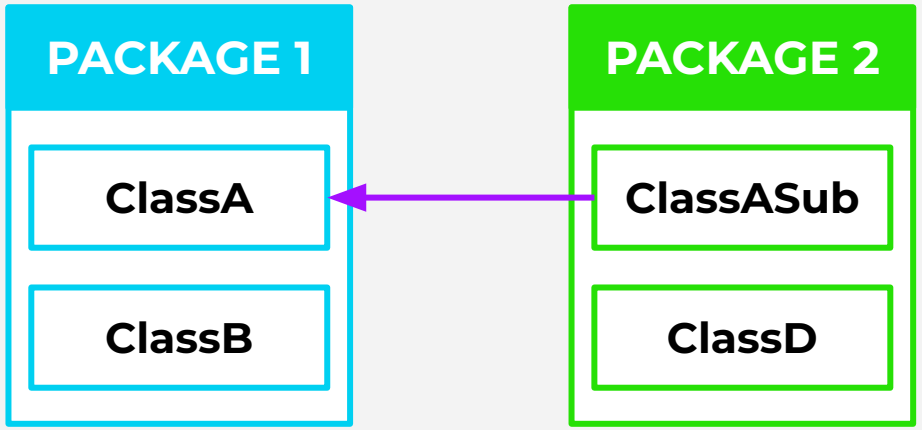


Can be accessed by **subclasses** from **any package**

Class Access Modifiers – Public

- Methods or variables declared as public are accessible from everywhere in the program
- The public access modifier has the widest scope among all other access modifiers

A **PUBLIC** METHOD OR PROPERTY IN **ClassA...**

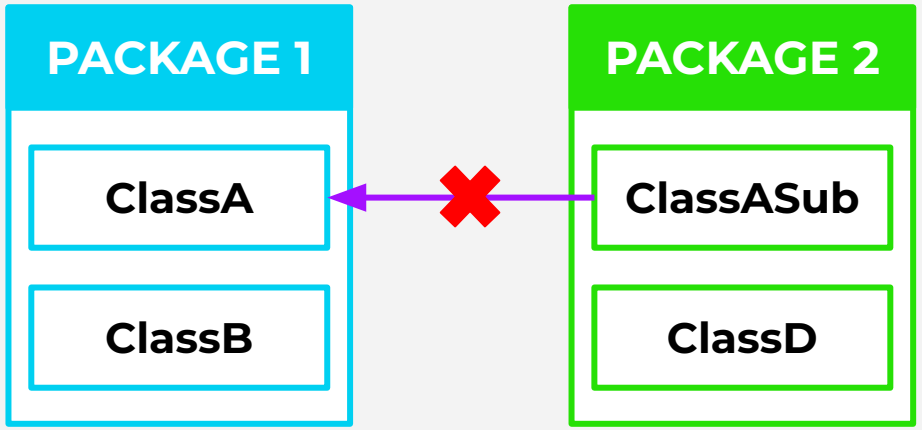


Can be accessed by **any class** from **any package**

Class Access Modifiers – Default

- The default access modifier is used whenever no access modifier is specified
- Methods or variables having a default access modifier are accessible only within the same package (package-private)

A METHOD OR PROPERTY WITH
NO ACCESS MODIFIER...



Can be accessed by **any class** from **Package 1**, but not from other packages.

Even subclasses from **other packages** do **not** have access to them

Questions so far?

Attribute and Method Access Modifiers

- There are four you'll need to keep in mind
  - Public
    - accessible by any class from any package
  - Private
    - Only accessible within the class they're declared in
  - Protected
    - Only accessible by subclasses from any package
  - Default (*no access modifier*)
    - Only accessible to classes from their package

Assuming these files are in the same package

```
public class Tweet {
    private int tweetId;
    public String caption;

    public void setTweetId(int i) {
        this.tweetId = i;
    }
}
```

```
public class Controller {
    private int tweetId;
    public String caption;

    public static void main(String[] args) {
        Tweet tweet = new Tweet();
        tweet.tweetId = 193325834;
    }
}
```

Is this allowed?

No. tweetId is private to the Tweet class.

BTW, in case you're wondering…

```
public class Tweet {
    private int tweetId;
    public String caption;
```

As an example…

```
    public void setTweetId(int i) {
        this.tweetId = i;
    }
```

vs

```
    public void setTweetId(int tweetId) {
        this.tweetId = tweetId;
    }

}
```

The keyword "this" refers to the instantiated object. So one might:

Access variables – `this.varName`

Call methods – `this.methodName()`

This convention is useful when you have conflicting variables in the same scope.

The 2nd method is syntactically correct

Assuming these files are in the same package

```java
public class Tweet {
    private int tweetId;
    public String caption;

    public void setTweetId(int i) {
        this.tweetId = i;
    }
}
```

```java
public class Controller {
    private int tweetId;
    public String caption;

    public static void main(String[] args) {
        Tweet tweet = new Tweet();
        tweet.setTweetId(193325834);
    }
}
```

Is this allowed?

Yes as setTweetId is public.

Assuming these files are in the same package

```
public class Tweet {
    private int tweetId;
    public String caption;

    public void setTweetId(int i) {
        this.tweetId = i;
    }
}
```

```
public class Controller {
    private int tweetId;
    public String caption;

    public static void main(String[] args) {
        Tweet tweet = new Tweet();
        tweet.caption = "Hello world";
    }
}
```

Is this allowed?

Yes… but this is discouraged. We usually want to avoid allowing others to modify an objects properties directly.

Encapsulation

- Is one of the four OOP Principles
- Objects are the most important units in an OOP system and must maintain their integrity
- Individual objects are responsible for managing (i.e. storing, updating and returning) their own object data.
- Other objects must not be able to read or edit data stored in other objects directly
- These must be done through access points, normally in the form of getter and setter methods.

Encapsulation: Good Practice

- In most cases, properties/variables should be kept private to a class
- To edit or retrieve properties, getter/setter methods should be used
  - Getter/setter methods give the class control over how data is shown or edited by other entities
  - Getters -> getVarName(); returns the variable
  - Setters -> setVarName(Var var); usually no return value

Note: For those using IDEs, there's usually a way to generate your getters and setter

Encapsulation: Good Practice

- Note that you don’t have to create getters and setters for all as class’s variables
- ***General rule of thumb:*** Allow the class to protect its own data

```java
public class Tweet {
    private Location location;

    public void setLocation(Location l) throws IllegalArgumentException {
        if(l == null) {
            throw new IllegalArgumentException();
        else {
            this.location = l;
    }
}
```

In this example, you might want to require that a Tweet always have a location (i.e. Location is non-null).

```java
public class Controller {
    private int tweetId;
    public String caption;

    public static void main(String[] args) {
        Tweet tweet = new Tweet();
        tweet.setLocation(null);
    }
}
```

This way, your Tweet protects itself – showing the importance of encapsulation.

Constructors

- When we create our objects, it makes sense to assign values upon creation – not create then set
- Hence, we can utilize Constructors or special methods that are only called during instantiation

Writing a Constructor

```
public class Person {

    public Person() {



    }

}
```

Has to match

No return type
(not even void)

Writing a Constructor

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

    public Person() {




    }

}
```

Writing a Constructor

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

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

    }

}
```

Constructor Usage

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

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

    }

}
```

This is a syntax error

```
public class Controller {
    public static void main(String[] args) {

        Person j = new Person();
        j.setName("Juan");
        j.setAge(17);

        Person m = new Person();
        m.setName("Mario");
        m.setAge(18);

    }
}
```

Constructor Usage

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

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

    }

}
```

This now adheres to the class's constructor

```
public class Controller {
    public static void main(String[] args) {

    Person j = new Person("Juan", 17);
    Person m = new Person("Mario", 18);
    }
}
```

Multiple Constructors

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

    public Person() {
    this.name = "No name";
    this.age = 0;
    }

    public Person(String name) {
    this.name = name;
    this.age = 0;
    }

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

Sets default values on creation

Requires the name variable

Requires the both variables

No Constructor…

If this is written…

```
public class Person {
    private String name;
    private int age;
}
```

Then the constructor is blank and requires no parameters.

It would look something like…

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

    public Person() {

    }
}
```

Questions? ☺️

Summary

- A class is a template, while an object is an instance
- There are access modifiers at the ***class-level*** and the ***attribute-/method-level***
  - Public
  - Private
  - Default
  - Protected (<u>for property-/method-level only</u>)
- Encapsulation advocates for protecting a class's data

Summary

- Getters and setters help with enforcing encapsulation
- Constructors help in initializing an object

Reading Assignment

- Kindly read up on Unified Modeling Language or UML
  - ***Reading Assignment 2***
- We'll discuss more of this next meeting and have an exercise as well ☺️

Keep learning…