Download

Day 10: OOP 1

Download

Agenda

- Reminder of Today
- Object Oriented Programming Recap
- Class
- Field Variables/Members
- Constructor
- Methods
- This
- Quiz 4
- In Class

Download

Reminder for today!

Reminder for today's lecture. Watch this lecture before class. We'll do the Quiz at the start of class, you'll then leave and come back for your allocated timeslot for the Project 1 Code Review.

Download

OOP

Download

Reminder that Classes and Objects are the heart of OOP.

We talked about Encapsulation, Abstraction, Polymorphism, and Inheritance a little last week.

Download

4 Pillars Recap

Download

Abstraction

The ability to take code that has similar functionality and move it into a more generalized implementation.

Download

Encapsulation

The ability to hide and store data within some class or object. This is heavily shown using the `private` and `public` keywords.

Data is hidden, with methods exposing and manipulating the data.

Download

Inheritance

The ability for classes to inherit data from parent classes using the extends and super keywords.

Download

Polymorphism

The ability to have different implementations for methods/functions, with different function parameters.

Download

Classes and Objects

Download

Classes

The structure that defines data and how methods interact with each other and with the inner data.

Download

Objects

Actual instance or copies from the class (blueprint).

Actual "thing" we will use.

uses the class as the structure, while the object is the instance we use.

Download

Class

Let's look at the structure of a class now!

Download

Structure

Download

A class follows the structure to the right:

First thing to note: the name of the file must match the name of the class in order for our code to work.

`ClassName.java`

```
public class ClassName {
  // field variables/members
  public ClassName(params) {
    // Constructor
  }
  // Methods
}
```

Download

Field Variables/Members

Download

In our classes, we can have variables that exist throughout the class. These are called field variables or members.

These variables belong to the class itself.

Download

Field Variables in Action

These variables are created as normal variables but we put them inside the class itself.

These are also preceded by either `private` or `public` or nothing at all.

```
public class User {

  private String name;

  public String id;

  byte age;
```

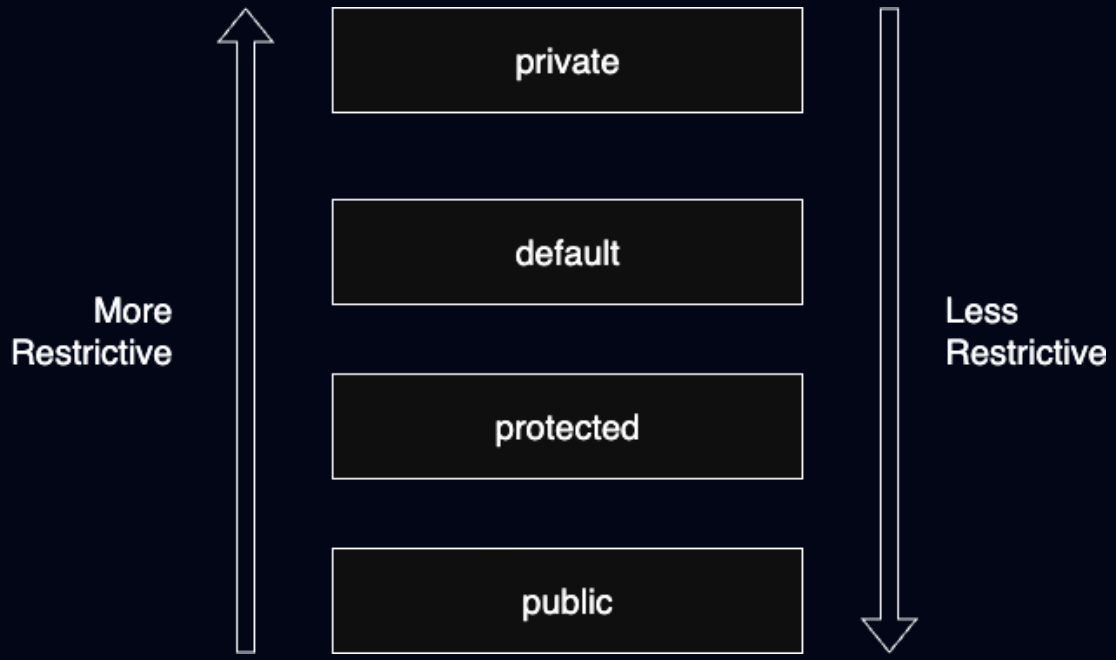
Field Variables in Action

In the example to the right, we have a **private** name for the user, a **public** id, and an age that has no public/private identifier. This means the age variable has a **default** scope.

```
public class User {

  private String name;

  public String id;

  byte age;
```

Download

Access Modifiers



Download

Modifier	Class	Package	Subclass	World
public	y	y	y	y
protected	y	y	y	n
default	y	y	n	n
private	y	n	n	n

Download

Field Variables in Action

This means our name is only visible in the class itself. Other classes can see the id. The age is visible to the class and technically to what we call the `package`

```
public class User {

  private String name;

  public String id;

  byte age;
```

Download

Packages?

Packages are a way we can group and organize code together. We may use packages later in the semester but for now, we will continue to write code without them.

Download

Constructor

Download

Our constructor is the method with the same name as our class. This is the method called when we create a new object of the class.

Download

In the example to the right, the constructor is the `public User(String newName)` function

```
public class User {
  private String name;
  public User(String newName) {
    name = newName;
  }
}
```

Download

When we create a new instance of the class with a new object, we're actually calling the constructor!

```
User u = new User("Edward");

// User("Edward") is calling the
User(String newName) method!
```

For example, when we created a new Scanner object, we actually were calling one of the constructors for the Scanner class!

```
Scanner input = new
Scanner(System.in);
```

Multiple Constructors

With Java, we can also create multiple constructors. These will vary with the function parameters and how the functions are implemented.

```
public User(String newName) {

  // implementation

}

public User(String fName, String
lName) {

  // implementation

}
```

Download

Multiple Constructors

This is actually the concept of **polymorphism** in action! We are able to have multiple forms for a single function!

```
public User(String newName) {

  name = newName;

}

public User(String fName, String
lName) {

  name = fName + lName;

}
```

Download

Methods

Download

Methods are what we call functions that belong inside a function. Methods are functions. However, not all functions are methods.

Technically, the static functions we wrote last week aren't methods as they don't belong to the object, but exist on the class itself. They don't use any data belonging to the object either.

Methods primarily use the data from the class itself and don't use the `static` keyword. This makes them **non-static** or also called **instance** methods.

Download

To write a method, we simply need the `public/private` keyword, followed by the return type, then the name of the method.

if we need parameters to the method, we can specify them in the parentheses.

```
public class User {
  private String name;
  public String getName() {
    return name;
  }
}
```

Download

For example with `getBirthYear`, where we take the current year as a parameter and return the difference of the current year and the stored age.

```
public class User {

  byte age;

  public int getBirthYear(int
  currYear) {

    return currYear – age;

  }

}
```

Download

Variable Scoping

Download

Class Scope

Field variables/members exist in the class itself.

This means any instance methods can see and use the variables.

```
public class User {
  private String name;
  // exist throughout the class
}
```

Download

Method Scope (params)

Parameters/variables defined from the method header in the parentheses only exist in the method itself.

In the example to the right, `currYear` only exists in the `getBirthYear` method.

```
public class User {

  public int getBirthYear(int
  currYear) {

    // currYear doesn't exist
    outside getBirthYear

  }

}
```

Download

Method Scope (local defined)

Variables defined in the methods, but not in the parameters, also only exist in the method themselves.

Values are returned from methods/functions, but not the variables themselves.

```
public class User {

  public int getBirthYear(int
  currYear) {

    int year = currYear - age;

    return year;

    // year only exists in the
    method, but the value leaves the
    function.

  }

}
```

Download

this

Download

In Java, if we want to reference anything in the class, within any method, we can use the `this` keyword.

Any method or variable within the class is accessible using the `this` keyword.

This is useful when we need to specify which variable we are referencing.

In the example the right, `name` appears as both a field variable **and** as a parameter to the constructor.

To avoid confusion, using the `this` keyword can help us differentiate between the two.

```
public class User {
  private String name;
  public User (String name) {
    this.name = name;
  }
}
```

Concurrently, this can help us call another constructor too!

If we use `this` as a method, it naturally will call another constructor, in this case, calling the constructor that takes in a single String.

```
public class User {

  public User(String name) {

    this.name = name;

  }

  public User(String fName, String
  lName) {

    this(fName + " " + lName);

  }

}
```

Download

Example Class

Download

```
public class Stationary {
    private String name;
    private String description;
    public Stationary(String name, String description) {
        this.name = name;
        this.description = description;
    }
    public String getName() {
        return this.name;
    }
}
```

Download

Calling methods

Download

In order to call a method, we must call it on the variable itself.

```
User ed = new User("Edward");

System.out.println(ed.getName())

// gives us "Edward"


User amy = new User("Amy");

System.out.println(amy.getName())

// gives us "Amy"
```

Download

Since these methods aren't static, they must be called on the objects. If we call them on the class, our code would fail to compile.

```
User u = new User("Edward");

System.out.println(User.getName());

// crashes due to getName not being static.
```

```
Error:

non-static method getName() cannot be referenced from a static context

User.getName();

^_________^
```

Download

Alright that's it for the lecture/content portion. Please come to class to do Quiz 4. We will do the Code Reviews afterwards. Please come on time for your slot. If you're late, it'll be skipped and you'll receive a 0 for the Code Review portion, impacting your grade heavily. While other students are doing the Code Review, please work on the in class.