

# COMP2511

---

## WEEK 2

Inheritance!!!

# A G E N D A

---

- Access Modifiers & Packages
- Code Review (super vs this)
- Basic Inheritance & Polymorphism
- equals

# ACCESS MODIFIERS & PACKAGES

---

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

# INHERITANCE

---

In Java, a class can inherit attributes and methods from another class. The class that inherits the properties is known as the sub-class or the child class. The class from which the properties are inherited is known as the superclass or the parent class.

Known as a "is-a" relationship

# SUPER VS THIS

---

```
public class Shape {
    public String color;

    public Shape(String color) {
        System.out.println("Inside Shape constructor");
        this.color = color;
    }
}
```

```
public class Rectangle extends Shape {
    public int height;
    public int width;

    public Rectangle(String color) {
        super(color);
        System.out.println("Inside Rectangle constructor with one argument");
    }

    public Rectangle(String name, int width, int height) {
        this(name);
        this.width = width;
        this.height = height;
        System.out.println("Inside Rectangle constructor with three arguments");
    }

    public static void main(String[] args) {
        Rectangle r = new Rectangle("red", 10, 20);
    }
}
```

# SUPER VS THIS

---

1. What is the difference between super and this?
2. What about super(...) and this(...)?
3. What are static fields and methods?

# SUPER VS THIS

---

## 1. What is the difference between super and this?

- `super` refers to the immediate parent class whereas `this` refers to the current class

## 2. What about `super(...)` and `this(...)`?

- **`super()`** acts as a parent class constructor and should be the first line in a child class constructor
- **`this()`** acts as a current class constructor (can be used for method overloading)

# SUPER VS THIS

---

## 3. What are static fields and methods?

- **Static fields and methods** are variables that are common and available to all instances of a Class. They belong to the Class, rather than an instance. There is only one copy for all instances.
- **Methods** are a block of code that perform a task. You can think of them as functions of a class.



# DOCUMENTATION

---

- Why is documentation important? When should you use it
- What does the term "self-documenting" code mean?
- When can comments be bad (code smell)?

# DOCUMENTATION

---

- Why is documentation important? When should you use it
- What does the term "self-documenting" code mean?
  - Code that documents itself. It is readable inherently. Usually accomplished through variable name and function names
- When can comments be bad (code smell)?
  - Comments become stale & does not get updated with new changes
  - Possibly hinting that your design/code is too complex

# DOCUMENTATION

---

```
// Single line comment
```

```
/**  
 * This is multi-line  
 * documentation  
 */
```

```
/**  
 * Constructor used to create a file  
 * @param fileName the name of the file  
 * @param content contents of the file  
 */
```

## CODE DEMO

---

1. Create a **Employee** class with a **name** and **salary**
2. Create setters & getters with JavaDoc
3. Create a **Manager** class that inherits **Employee** with a **hireDate**
4. Override **toString()** method
5. Write **equals()** method

# HOW MANY CONSTRUCTORS DOES A CLASS NEED?

---

Technically none. If a class is defined without a constructor, Java adds a default constructor.

However, if a class needs attributes to be assigned (e.g., has a salary), then a constructor must be assigned.

If your class has attributes with no default values, then the **constructor must set these attributes**. This is because variables with no values are dangerous (null), and is also the constructor responsibility.

Each class's constructor is also only responsible for setting **its own attributes**. **Do not set the superclass's attributes within the subclasses without using a super(...) constructor call.** (In fact super is implicitly called for all subclasses)

## `toString()`

---

The [documentation](#) for the `toString()` method states that it should return a string that "textually represents" the object. In this case, it should contain the name, salary and hire date (in the case of `Manager`), but also the runtime class of the object.

# EQUALS - MOTIVATION

---

- What does the '==' operator do when comparing objects?
- Where have you seen this sort of behaviour before in other languages? How is the underlying data checked for equality in that scenario?
- How can we compare two objects for equality?

# EQUALS

---

- Since we are overriding an existing method (in the super most class called Object), we must follow the conditions described.
- The conditions can be found in the [Java Docs](#)



# EQUALS

---

Typical Structure of the equals method will include:

- Check that the passed in object is not null. if (object == null) return false
- Check if the passed in object is the same instance as the calling object.
- if (this == object) return true
- Check the concrete type of the calling object matches the concrete type of the passed in object.
- if (!this.getClass().equals(object.getClass())) return false
- Typecast and then check if all fields are equal

# EQUALS - WITH INHERITANCE

---

What is the relationship between a super type and a sub type in terms of equality? Can a concrete instance of an Employee be equal to an instance of a Manager

- No because of the `.getClass()` call which will check and compare that the classes are indeed different.

# EQUALS - WITH INHERITANCE

---

How can the method in Manager utilise code in Employee to avoid repetition?

- Obviously, we can use the super method!!!

# LABBING :eyes:

---

**B R R R R R**