# RE-BOOTCAMP

OOP

# OBJECT ORIENTED PROGRAMMING CONCEPTS

- ❑ Encapsulation
- ❑ Inheritance
- ❑ Abstraction
- ❑ Polymorphism

# ACCESS MODIFIERS

- Access modifiers can be applied to variables, classes, methods, constructors

Public - accessible in the whole project

Private - accessible only within the same class

Protected - accessible only in the same package or sub classes in other packages (inheritance)

Default - accessible only in the same package. If no modifier is given this is the default modifier

Public > Protected > Default > Private

# ENCAPSULATION

- Encapsulation allows the programmer to hide and restrict access to data. More specifically encapsulation will allow an object to hide its variables and methods from being accessed outside of a class
- To achieve encapsulation:
    1) Declare the variables with private access modifier
    2) Create public getters and setters that allow access to those variables

```java
public class Person {

    private String name;
    private int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

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

    Person p1 = new Person();

    p1.setName("Mike");
    p1.setAge(27);

    System.out.println("Name: " + p1.getName() +
                       "\nAge: " + p1.getAge());

}
```

# NOTE

We can provide only getter in a class to make the class immutable. (Read only)

We can provide only setter in a class to make the class attribute write-only.

# EXAMPLE

In my project I created multiple POJO/BEAN classes in order to manage test data and actual data.

I take JSON from API response and convert to object of my POJO class. All the variables are private with getters and setter. Also in order to store credentials or sensitive data in my framework I have use encapsulation, configuration reader also known as property file or excel sheet to hide data from the outside. I use Apache POI if the data store in Excel in order to extract/read and modify data.

# INHERITANCE

- Inheritance allows one class to inherit content from another class (fields and methods), or in other words, allows a class to be a copy of another class
    - Allowing code reusability

- Super class (also known as parent or base class): is the class where the fields and methods that are being inherited or passed on
- Sub class (also known as the child or derived class): is the class getting the inherited fields and methods

# INHERITANCE

- In order to use inheritance in java we use keyword extends

-> In this example Dog is the subclass

and Animal is the super class

```
public class Dog extends Animal {

}
```

Note: One class cannot have more than one super class

Note: Every class in java inherits Object class

# INHERITANCE - WHAT IS INHERITED?

- All public variables and methods
- All protected variables and methods
- All default variables and methods if the super and sub classes are in the same package

---

## What is not Inherited:

- Constructors are not inherited
- Any private variables or methods are not inherited

# EXAMPLES

```
public class A{

}

public class B extends A{

}
```

Single inheritance

```
public class A{

}

public class B extends A{

}

public class C extends B{

}
```

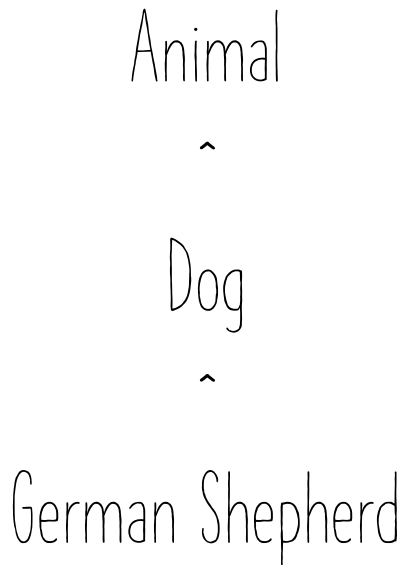
Multi-level inheritance

```
public class A{

}

public class B extends A{

}

public class C extends A{

}

public class D extends A{

}
```

Hierarchical inheritance

# IS-A RELATIONSHIP

- IS-A relationship can be used to describe the relationship between classes that are using inheritance
- Ex: In this example we can say:
  - German Shepherd is a Dog
  - Dog is a Animal
  - German Shepherd is a Animal



# SUPER KEYWORD

The super keyword in java is a reference variable that is used to refer to parent class objects in inheritance

- Super with variables (super.variable)
- Super with methods (super.method)
- Super with constructors (super() )

# INHERENCE - CONSTRUCTORS

- The super class constructor always executed before the sub class constructor
- It is possible to call the super class' constructor using keyword super
  - The default no parameter constructor automatically invokes its super class constructor
- Rules:
  - If this super constructor is called explicitly it must be the first line in the sub class constructor: super()
  - The parameters must match with parent constructor. If parent class has only constructors with parameters, then an explicit call must be made matching those parameters

# INHERITANCE - STATIC MEMBERS

- If the access modifier allows inheritance, static members will also be inherited
- Static variables will be shared for all objects related to that class. All the object from the parent to the child
- Static methods can be called using the parent class name or sub class name

# OVERRIDING METHODS

- After inheriting a method from the parent class the action that the child class may need to perform could be different, so we are able to keep the same method, but change the implementation to better match the child class' needs
- We can use the @Override annotation before the method to tell the compiler we are overriding a method. It will help ensure the method is overridden correctly.
- Ex: toString() method

# OVERRIDING METHOD RULES

1. There must be is-a relationship (inheritance)
2. The method must have the same name as in the parent class
3. The method must have the same parameter as in the parent class
4. Access modifier: Needs to be same or more visible
5. Return type:
    - must be same or
    - covariant type (same class type or sub class type)

# OVERRIDING STATIC METHODS

- You cannot override static methods. If you try, they will become hidden
- A hidden method occurs when a child class defines a static method with the same name and signature as a static method defined in a parent class.
- Method hiding is similar but not the same as method overriding.
- The rules for overriding a method are the to hide a method, but in addition, the method must stay static

| Method Overloading | Method Overriding |
| --- | --- |
| Method overloading is performed within class | Method overriding occurs in two classes that have IS-A relationship |
| Parameters must be different | Parameters must be same |
| Access specifier can be changed | Access specifier cannot be more restrictive than original method |
| private and final methods can be overloaded | private and final methods can not be overridden |
| Return type of method does not matter in case of method overloading, it can be same or different | Return type must be same or covariant in method overriding |

# INHERITED VARIABLES

- Unlike methods, it is not possible to override a variable, but if it is possible to hide them. Variable hiding occurs when a variable is declared with the same name as a variable from the super class
- After hiding a variable both are still accessible. The super variable can be accessed with the super reference and the new variable can be accessed with the child reference

# THIS VS SUPER

- super is used to access/call the parent class members (variables and methods)
- this is used to call the current class members (variables and methods).
  - Can be used when parameter local variables have the same name is instance variables
- Both can be used anywhere in a class, except static block or static method

# THIS() VS SUPER()

- this() is used to call an overloaded constructor in the same class.
- super() is used to call the constructor from the parent class
- Both this()  and super() must be the first statement in the constructor. This result in one constructor calling another constructor.

# INHERITANCE EXAMPLE

In my framework I have a TestBase class where I store all my reusable code and methods. My test execution classes, and elements classes will extend the TestBase in order to reuse the code.

My framework follow POM and some pages have similar actions, so I can easily use those similar actions and fields by inheriting some a common class

# QUESTIONS - WHAT IS OVERRIDING?

Overriding means changing the implementation of a method that is coming via inheritance from a super class

Example: get method

WebDriver driver = new ChromeDriver();

driver.get("URL") ==> opens the url from chrome

WebDriver driver = new FireFoxDriver();

driver.get("URL") ==> opens the url from Firefox

# TASK

1. Create a class Book with instance variables: title, type, author, price
2. Create a sub class: Ebook with additional fields: size, pages and a readBook method
3. Create a sub class: AudioBook with additional fields: length, narrator and listen method

# TASK

- Create a class Shape. It will have int variables: x and y
- Create a constructor that will initialize those values

- Create a Rectangle and inherits Shape class
- Add two more variables: width and height
- Create a constructor that accepts all 4 fields and initializes them
- Overload the constructor to only accept x and y and set 0 as the default for width and height