RE-BOOTCAMP

OOP

OBJECT ORIENTED PROGRAMMING CONCEPTS

- [x] ~~Encapsulation~~
- [x] ~~Inheritance~~
- [ ] Abstraction
- [ ] Polymorphism

ABSTRACTION

- A concept that focuses only on the feature / idea of something and not how it is done (implementation)
- In java, abstraction is achieved via abstract class or interface
- Abstraction cannot exist without inheritance

ABSTRACT CLASS

- To create an abstract class: add keyword abstract before the class declaration
- An abstract class cannot be instantiated - An object can never be created from that class
- An abstract class allows creation of abstract methods, which are declaration of method signatures without any implementation. In other words, methods without a body.
- The class which inherits the abstract class is called the concrete class. The concrete class must implement all abstract methods declared from the abstract class
- Variables act following regular inheritance rules

ABSTRACT CLASS EXAMPLE

Abstract class called Student →

Abstract method named attendClass() →

Concrete class named LocalStudent →

Implementation of abstract method attendClass →

```
public abstract class Student{

    public abstract void attendClass();

}


public class LocalStudent extends Student{

    @Override
    public void attendClass(){

        System.out.println("Attending in person");

    }

}
```

EXTEND TO ANOTHER ABSTRACT CLASS

- It's possible for an abstract class to inherit another abstract class. In this case implementation of the abstract methods are not required
- In this case the sub abstract class is not a concrete class, but as soon as a concrete class inherits all the properties of that abstract class it must implement all abstract methods from all super classes

```
public abstract class Student {

    public abstract void study();

}

public abstract class LocalStudent extends Student {

}

public class CollegeStudent extends LocalStudent{

    @Override
    public void study(){
       System.out.print("College student studying");
    }

}
```

ABSTRACT CLASS RULES RECAP

- Abstract classes can not be instantiated
- Abstract classes may be defined with any number of abstract and non-abstract methods, including none
- Abstract classes can not be private or final
- An abstract class that extends another abstract class inherits all its abstract methods as its own abstract methods
- The first concrete class that extends an abstract class must provide an implementation for all the inherited abstract methods

INTERFACE

- The second way to achieve abstraction in java is via interface
- An interface is not a class, but acts similar
- An Interfaces also allow creation of abstract methods
- The main purpose of an interface is providing additional information and behaviors to any class that needs it
- To create an interface the keyword interface is used instead of class

```
public interface Teachable{


}
```

INTERFACE CONT...

```java
public interface Teachable {

    public abstract void doHomework();
    // can be written just:
    // void doHomework();
}
```

- By default any method created in an interface is < public abstract >
- [Java 8] Only way to create non-abstract methods in an interface is to create a default or static method
    - Default methods: use keyword default before the return type of a method. This will allow a method to be created with implementation.
    - Static methods: use keyword static before the return type of the method. This will allow a method to be created with implementation.

```java
public interface Teachable {

    public static void study(){
        // implementation
    }

}
```

```java
public interface Teachable {

    public default void learn(){
        // implementation
    }

}
```

INTERFACE CONT...

- By default any variable created will be < static final > and must be initialized
- To add an interface to a class, keyword implements is used after the class name
- Inheritance allows only parent class, but it is possible to implement multiple interfaces to a class
    - This is done by adding a comma after each interface after the class declaration. The class must implement all the abstract methods from all the interfaces
- It's possible to extend interfaces to other interfaces. And unlike regular inheritance interfaces can extend as many interfaces as they wish

```
interface Upgradeable extends Teachable, Movable {

}
```

```java
interface Moveable {

    void rotate();

}

interface Upgradeable{

    void increaseSize();

}

public class TV implements Moveable, Upgradeable {

    public void rotate(){
        //
    }

    public void increaseSize() {
        //
    }
}
```

INTERFACE CONT...

- Is-A relationship: interface implementation is included.

```
public class Student extends Person implements Teachable,Dreamer{


}
```

In this case: Student is a Person, Student is a Teachable, Student is a Dreamer

 Note: Here we inherit a Person class and implement Teachable and Dreamer interfaces

Interfaces can have:

- Constant variables
- Abstract methods
- Default Methods
- Static Methods

Interfaces can not have:

- Constructor
- Blocks
- Instance variables or methods

ABSTRACT METHOD RULES RECAP

- Abstract methods can only be defined in abstract classes or interfaces
- Abstract methods can not be declared private or final
- Abstract methods can not provide a method body/implementation in the abstract class or interface it is declared
- Implementing an abstract method in a subclass follows the same rules for overriding a method

ABSTRACTION QUESTIONS

- Do abstract classes have a constructor?

  -> Yes, the class is still inheriting the Object class and has a default constructor

- Can an abstract class have only non-abstract methods?

  -> Yes, an abstract class doesn't need to have any abstract methods.

- If the abstract class doesn't have any abstract methods can you instantiate?

  -> No, objects of an abstract classes can never be made

ABSTRACTION QUESTIONS

- Can you add instance or static variables into abstract class?

  -> Yes

- Can you add initializer or static blocks into Abstract class?

  -> Yes

- What access modifiers can abstract methods be?

  -> They can have any access modifier except private

ABSTRACTION QUESTIONS

- Can an abstract method be static?

  -> No, abstract methods are meant to overridden, and only the instance methods can be override

- Can a method be abstract and final?

  -> No, we cannot have abstract final methods. Final methods cannot be overridden, and abstract methods must be implemented

ABSTRACTION EXAMPLE

- In my framework I have achieve abstraction by using collections or Map, because it's all interface. Most of the cases I come across using List. If we want to access elements frequently by using index, List is a way to go. ArrayList provides faster access if we know index. If we want to store elements and want them to maintain an order, List is a better choice.

  i) List<String> webs = driver.getWindowHandles();
  =>create a list first to store web URLs in list

  ii) findElements evaluates multiple elements so therefore will assigned to List <WebElement>

  iii) To handle dynamic elements store it in the list and identify by index:
  List<WebElement> all = driver.findElements(By.tagname(""));

ABSTRACTION EXAMPLE

- In my framework I follow POM and had situations where some pages shared similar actions that were similar but worked slightly different, so I was able to use abstraction to define those actions and implement them in each page according to what was needed for that webpage

```
Object obj = new Object();
  [1]    [2]    [3]    [4]

  [1] reference type
  [2] reference name
  [1 + 2] reference

  [3] new keyword
  [4] constructor call
  [3 + 4] object
```

OBJECTS REVIEW

POLYMORPHISM

- Polymorphism means to take many forms
- In java Polymorphism is the ability of an object to take many forms. This is done through the reference of an object
- Three reference types an object can have:
    1. Itself
    2. Super classes
    3. Interfaces which are implemented

POLYMORPHISM CONT...

```
public class Animal{

}

interface Tameable {

}

public class Dog extends Animal implements Tameable{

}
```

Possible Dog object references:

```
Dog dog = new Dog();
Animal dog = new Dog();
Tameable dog = new Dog();
```

POLYMORPHISM CONT...

- The reference of an object will determine which class members the object can access
- If a method is called the object will use the method implementation of the object type. If the method is not overridden in the sub class, the super classes method implementation is executed
- The object will always give the values of the object itself if the reference of the object also has access to that class member. If there is a class member that is out of scope of the reference, then there will be a compile time error.

REFERENCE CASTING

- Reference casting is casting the reference to another reference of the object. Casting does not happen between objects.
- Down casting: Going from a super class reference to sub class reference. This must be done explicitly.
- Up casting: Going from a sub class reference to a super class reference. This is done implicitly.

- The reference type is giving in parentheses next to an object to cast the reference to that reference type.

POLYMORPHISM CONT...

- Static polymorphism is method overloading (Compile time)
- Dynamic polymorphism is method overriding (Runtime)

- One use of polymorphism is for polymorphic collections. Being able to store different types of objects that share similar reference types into one data structure

EXAMPLE

WebDriver driver = new ChromeDriver();

=>we are initializing Chrome browser using Selenium WebDriver. It also means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object (from ChromeDriver class).

- Will cover more real examples during Collections next session