

COMP 250

INTRODUCTORY SCIENCE

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Week 3-1: OODS Interf

Add WeChat: edu_assist_pro

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Interfaces (disclaimer: we'll talk about interfaces pre Java 8)

Assignment Project Exam Help

- Generics

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

INTERFACES

- `interface` is a reserved keyword in Java.

- Like classes, interfaces can be declared to be public or package-private.

<https://eduassistpro.github.io/>

- Similarly to classes, interfaces can have methods but the following restrictions apply:

Add WeChat edu_assist_pro

- All methods are by default public and abstract.
 - All fields are by default public, static, and final.
- Interfaces cannot be instantiated.

SYNTAX

- We declare an interface using the `interface` keyword.

Assignment Project Exam Help

```
public interface {  
    :  
}
```

<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

- An interface is implicitly abstract. You do not need to use the `abstract` keyword while declaring an interface.

EXAMPLE

```
public interface MonsterLike {  
    public  
    public  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- The methods are all implicitly abstract.

INHERITANCE

- To use an interface you first need a class that *implements* it. Interfaces specify what a class must do and not how. It is the blueprint of the class.

Assignment Project Exam Help

- A class can *implement* one or more interfaces. The keyword `implements` is used to achieve this. Interfaces are used to achieve this.

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- If a class implements an interface and does not implement all methods specified by the interface then that class must be declared `abstract`.
- It is possible for a Java interface to *extend* another Java interface, just like classes *extend* other classes. You specify inheritance using the `extends` keyword.

IMPLEMENTS

```
public class Dragon implements MonsterLike {  
    :  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- **Inside the class Dragon, the methods `sp` `unAway()` must be implemented!**
- **Note:** if the interfaces are not located in the same packages as the implementing class, you will also need to import the interfaces. Java interfaces are imported using the import statements just like Java classes.

INTERFACE INSTANCES

- Once a Java class implements an Java interface you can use an instance of that class as an instance of that interface

Assignment Project Exam Help

```
public interface MonsterLike {
    public int spook();
    public void runAway();
}
```

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```
public class Orc implements MonsterLike {
    :
}
```

```
public class Dragon implements MonsterLike {
    :
}
```

```
class Hero {
    public double fight(MonsterLike m) {
    }
}
```

```
Hero frodo = new Hero();
MonsterLike thrall = new Orc();
Dragon drogon = new Dragon();
frodo.fight(thrall);
frodo.fight(drogon);
```


EXTENDS + IMPLEMENTS

Classes can extend at most one class, but they can implement multiple interfaces.

Example:

Assignment Project Exam Help

```
public class Dragon extends Enemy, MonsterLike, FireBreather {  
    :  
}
```

<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

Dragon is a subtype of (at least) Enemy, MonsterLike, and FireBreather. An instance of Dragon can be used whenever an object of those types is required.

INTERFACES VS ABSTRACT CLASSES

ABSTRACT CLASS

Not all methods have to be abstract.

The abstract keyword must be used to declare a class to be abstract.

Can contain methods that have been implemented as well as instance variables.

Abstract classes are useful when some general methods should be implemented and specialization behavior should be implemented by child classes.

INTERFACE

All methods are abstract by default.

Explicitly abstract.

No method can be implemented and only constants (final static fields) can be declared.

Interfaces are useful in a situation that all properties should be implemented.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

POST JAVA 8

From Java 8/9 onwards, interfaces can also contain the following

- Default methods **Assignment Project Exam Help**
- Static methods **<https://eduassistpro.github.io/>**
- Private methods **Add WeChat edu_assist_pro**
- Private Static methods

WORKING TOWARD GENERICS

- Suppose I'd like to create a class that defines a new type `Cage`. I would like to use this in a class called `Kernel` where I have a bunch of objects of type `Dog`.
- What if later on I also happen to need cages for objects of type `Bird`?
- Can I use the same class? Should I create a new class with the same features but where instead of `Dog` I use `Bird`? Is there a better solution?

```
public class Cage {  
    private Dog occupant;  
  
    public void lock(Dog p) {  
        is.occupant = p;  
    }  
  
    public void peek() {  
        rn this.occupant;  
    }  
  
    public void release() {  
        this.occupant = null;  
    }  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat: edu_assist_pro

GENERIC IN JAVA

- A generic type is a class or interface that is parameterized over types. We use angle brackets (<>) to specify the type parameter.

- Example →

```
public class Cage<T> {  
    private T occupant;  
  
    public void lock(T p) {  
        is.occupant = p;  
    }  
  
    public T peek() {  
        return this.occupant;  
    }  
  
    public void release() {  
        this.occupant = null;  
    }  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat [edu_assist_pro](#)

EXAMPLE – CAGE<>

We can now create cages containing different type of objects, depending on the need:

```
Cage<Dog> crate = new Cage<Dog>();  
// now inside crate we can lock only Dogs!  
Dog snoopy = new Dog();  
crate.lock(snoopy);  
  
Cage<Bird> birdcage = new Cage<Bird>();  
// if we call lock on birdcage we must provide a Bird as input.  
Bird tweety = new Bird();  
birdcage.lock(tweety);  
  
// peek() called on crate returns a Dog,  
// peek() called on birdcage returns a Bird!  
Dog d = crate.peek();  
Bird b = birdcage.peek();
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

GENERIC TYPE NAMING CONVENTIONS

- Java Generic Type Naming convention helps us understand code easily.
- Usually type parameter names are single uppercase letters to make it easily distinguishable from java variables. used type parameter names are:
 - E – Element
 - K – Key (Used in Map)
 - N – Number
 - T – Type
 - V – Value (Used in Map)
 - S, U, V etc. – 2nd, 3rd, 4th types
- More about generic type: <https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html>
<https://docs.oracle.com/javase/tutorial/java/generics/bounded.html>

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

BOUNDED TYPE

- Sometimes we might want to restrict the types that can be used to create a Cage (or a general parameterized type)
- We can do that using the `Context` extends is used to mean either "extends" (as in classes) or "`Context`" (as in interfaces)
- Not only this will limit the types we can use to instantiate a generic type, but it will also allow us to use methods defined in the bounds.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

BOUNDED TYPE

- Example:

```
public class Cage<T extends MonsterLike> {  
    private T occupant;  
  
    public  
        this.occupant.sp  
        return this.occu  
    }  
  
    public void release() {  
        this.occupant = null;  
        this.occupant.ranAway();  
    }  
}
```

EXAMPLE – LIST INTERFACE

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

EXAMPLE – LIST INTERFACE

```
public interface List<E> extends Collection<E>{  
    boolean add(E e);  
    void add(int i, E e);  
    boolean isEmpty();  
    E get(int i);  
    E remove(int i);  
    int size();  
    :  
}
```

Some of the methods are inherited from the interface Collection, while others are declared inside List.

EXAMPLE – LIST INTERFACE

The documentation explains exactly how each of these method should behave. For example:

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html#add-E->

EXAMPLE – LIST INTERFACE

The documentation explains exactly how each of these method should behave. For example:

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

EXAMPLE – ARRAYLIST

```
public class ArrayList<E> implements List<E>{
    boolean add(E e) {...}
    void add(int i, E e) {...}
    boolean isEmpty() {...}
    E get(int i) {...}
    E remove(int i) {...}
    int size() {...}
    void ensureCapacity(int i) {...}
    void trimToSize() {...}
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

All of the methods inherited from `List` are implemented. In addition, others are declared and implemented in `ArrayList`.

EXAMPLE – LINKEDLIST

```
public class LinkedList<E> implements List<E>{
    boolean add(E e) {...}
    void add(int i, E e) {...}
    boolean isEmpty() {...}
    E get(int i) {...}
    E remove(int i) {...}
    int size() {...}
    void addFirst(E e) {...}
    void addLast(E e) {...}
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

All of the methods inherited from `List` are implemented. In addition, others are declared and implemented in `LinkedList`.

HOW ARE INTERFACES USED?

```
List<String> greetings;
```

```
greetings = new ArrayList<>();
```

```
greetings.add("Hello");
```

```
:
```

```
greetings = new LinkedList<String>();
```

```
greetings.add("Good day!");
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Interfaces define new data types.

can create variables of those

and assign to them any value

encasing to instances of classes

that implement the specified
interface!

HOW ARE INTERFACES USED?

```
public void myMethod(List<String> list)
:
list.add("one more");
:
list.remove(3);
:
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Whenever an object of type `List` is required, any instance of any of classes that implement `List` e used.

this case, `myMethod()` can be called both with an `ArrayList` or a `LinkedList` as a parameter.

HOW ARE INTERFACES USED?

```
public void myMethod(List<String> list) {  
    :  
    list.add("one m  
    :  
    list.remove(3);  
    :  
    list.addLast("Bye bye"); // compile-time error. Why??  
}
```

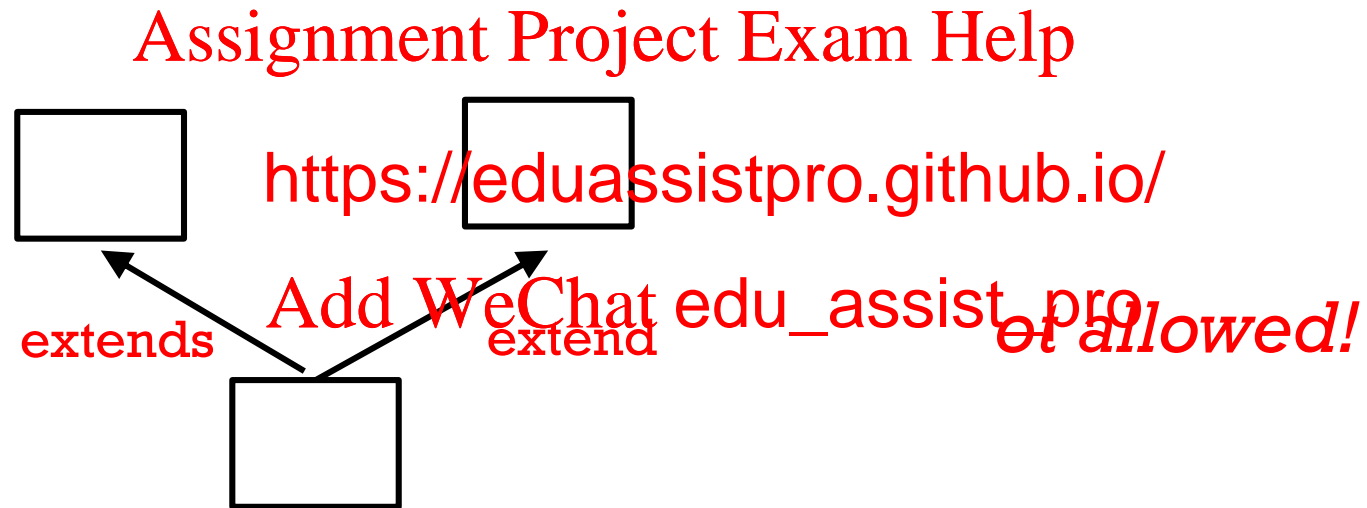
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

INHERITANCE

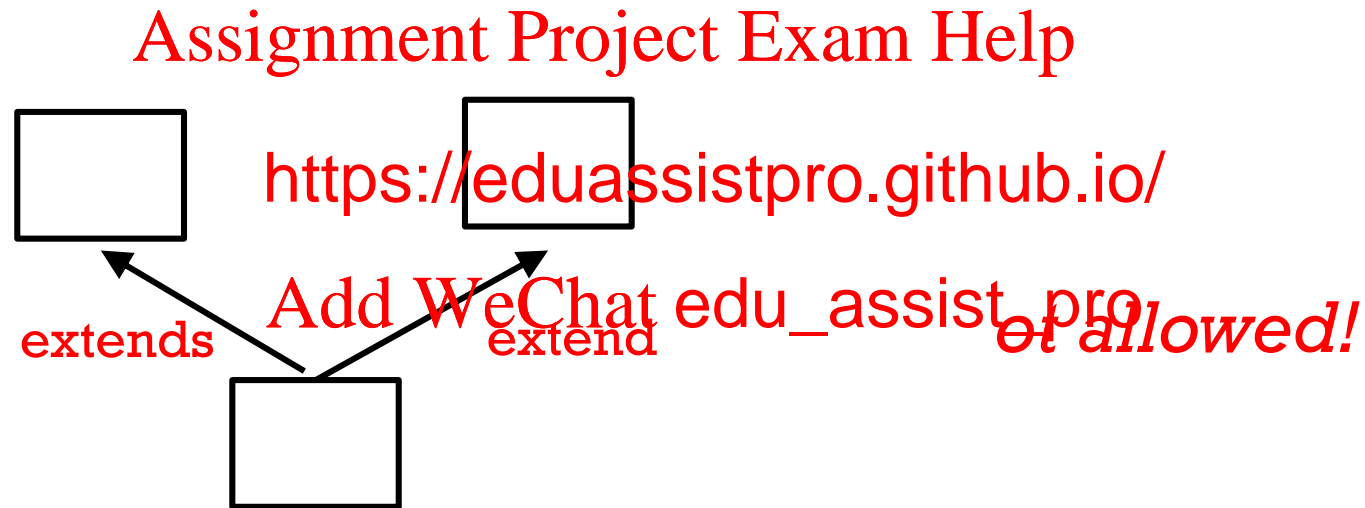
Remember that a class (abstract or not) cannot extend more than one class (abstract or not).



- Why not?

INHERITANCE

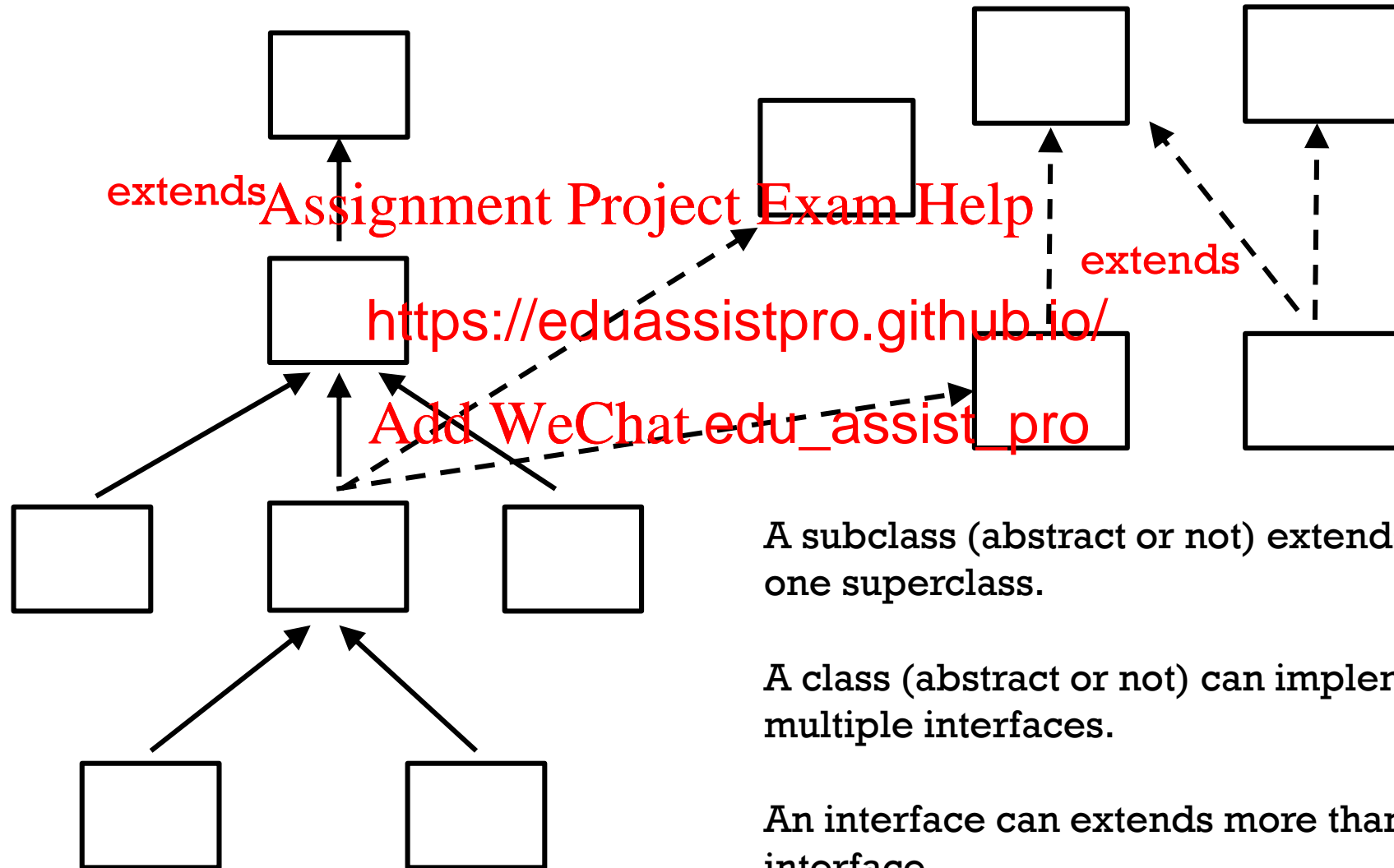
Remember that a class (abstract or not) cannot extend more than one class (abstract or not).



- Why not? *The problem could occur if two superclasses have implemented methods with the same signature. Which would be inherited by the subclass?*

classes (abstract or not)

interfaces



A subclass (abstract or not) extends exactly one superclass.

A class (abstract or not) can implement multiple interfaces.

An interface can extend more than one interface.

An orange paint roller with a red handle, positioned horizontally. The roller is partially filled with orange paint, and there are orange paint splatters and drips around it. The text "Coming Soon" is written in white on the orange surface of the roller.

Coming Soon

Assignment Project Exam Help

In the next

- Compar <https://eduassistpro.github.io/>
Add WeChat edu_assist_pro