Hey there, future Java maestro! You've come a long way, and it's time to bring all those skills together in a fun and exciting final project. Up to this point, you've nailed the essentials of Java and object-oriented programming. You learned how to make your code reusable with methods, got familiar with utility classes, and wrapped your head around the Main class. You explored inheritance, polymorphism, and encapsulation, setting the stage for designing scalable and maintainable code. Plus, you delved into logical program design, access modifiers, abstract classes, and interfaces— all the cool stuff that makes your code robust and efficient.

Now, let's get you started on an exciting course project: Creating a zoo! You will start small with a base Animal class and two animals: A tiger (which is a land animal) and a dolphin (which is a water animal). These animals will share common properties and functionality from the **`Animal`** class. By extending the **`Animal`** class, you'll also leverage the power of code reuse, ensuring you set up code to be reused as effectively as possible. Functionality will be enforced by implementing interfaces. The interfaces are implemented selectively to mirror the behavior of actual animals; for example, there is no reason to implement the walking functionality in a dolphin!

Goal

Utilize abstract classes, interfaces, and functionality of interfaces to help create a small zoo with a tiger and a dolphin, each inheriting common properties from an **`Animal`** class. You will extend the **`Animal`** class to create these unique animals. The **`Animal`** class implements an interface named **`Eat`**. The interface has two methods, but it implements only one. Since the **`Animal`** class does not implement all the methods of the **`Eat`** interface, the child classes will implement the unimplemented methods. You will also create two interfaces named **`Walk`** and **`Swim`** and apply the functionality of the methods of the **`Walk`** and **`Swim`** interfaces in certain classes only.

You will modify an interactive menu system to work with the objects of the classes that you create. Additionally, you will develop a multi-functional **`Penguin`** class that can both walk and swim. Finally, you'll test the entire program to ensure all functionalities work seamlessly.

Here is a quick breakdown of the tasks:

- Task 1 focuses on inheriting the common properties and functionality of the abstract class. Here you will create the two animal classes: the **`Tiger`** and **`Dolphin`** classes.
- Task 2 will step you through creating an interface, **`Swim`**, which declares the swimming functionality. It will also create the walking functionality in the **`Walk`**

interface. You will also utilize the **`Swim`** interface in the **`Dolphin`** class and the **`Walk`** interface in the **`Tiger`** class.

- Task 3 is about utilizing the menu system provided in the **`Main`** class to display the details of the appropriate animal the user requests.
- Task 4 involves creating another new class to represent a **`Penguin`** and implementing multiple interfaces so that the **`Penguin`** can both walk and swim.
- Task 5 is where you will finally test the functionality of your program.

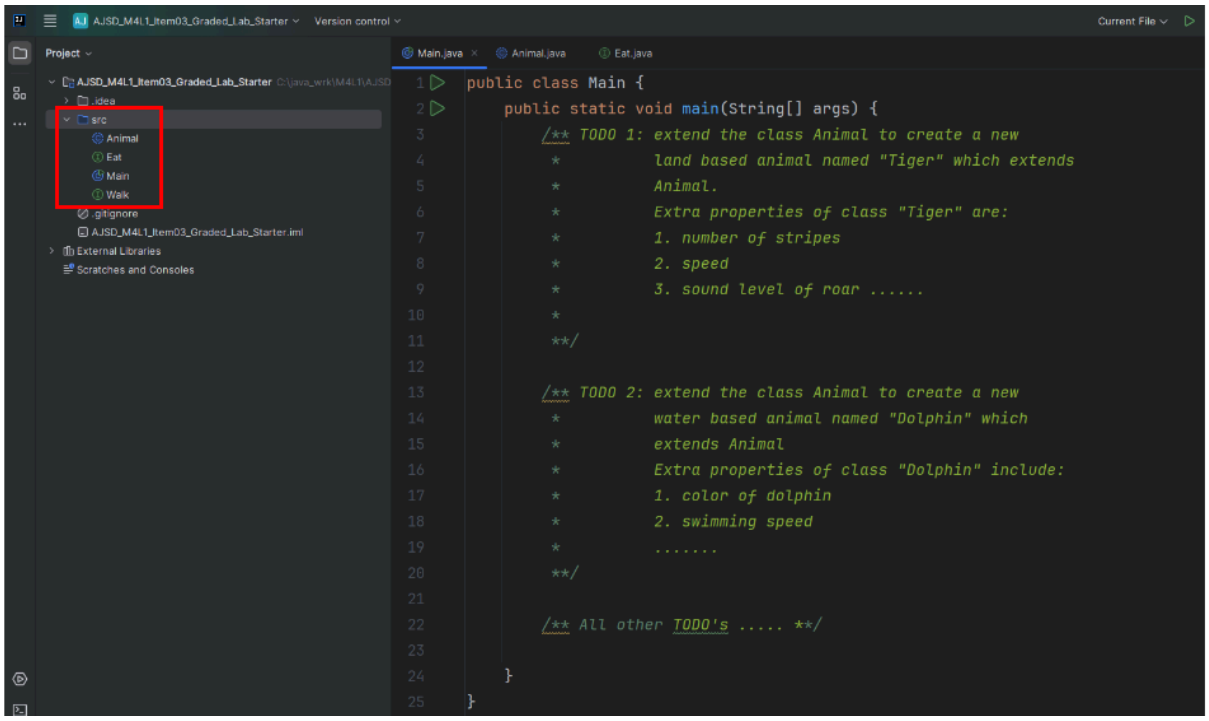
With all the skills you've picked up along the way, you're all set to create a program that's not just functional but also super elegant. This is your chance to show off your learning, creativity, and problem-solving prowess. Dive in, have fun, and bring your virtual zoo to life! Let’s get started!

🖥️ Note: When you encounter this icon, it’s time to get into your integrated development environment (IDE) and start coding!

In your lab environment, open IntelliJ by double-clicking on the icon on the desktop.

When the IntelliJ IDE opens, you will be brought to the starter code. The **`main`** method of the **`main`** class is placed in a file named *Main.java*. The starter code includes the following files:

- *Animal.java*
- *Eat.java*
- *Walk.java*
- *Main.java*



```java
public class Main {
    public static void main(String[] args) {
        /** TODO 1: extend the class Animal to create a new
         *          land based animal named "Tiger" which extends
         *          Animal.
         *          Extra properties of class "Tiger" are:
         *          1. number of stripes
         *          2. speed
         *          3. sound level of roar ......
         *
         **/

        /** TODO 2: extend the class Animal to create a new
         *          water based animal named "Dolphin" which
         *          extends Animal
         *          Extra properties of class "Dolphin" include:
         *          1. color of dolphin
         *          2. swimming speed
         *          .......
         **/

        /** All other TODO's ..... **/

    }
}
```

The base class of all the animals that we are going to create is the `Animal` class, and it has the following properties:

- private String `nameOfAnimal;`
- private int `weight;`
- private int `height;`
- private int `age;`

This class implements the `Eat` interface. The `Eat` interface has two methods:

- `eatingFood()`
- `eatingCompleted()`

The `Animal` class only implemented one of these methods, `eatingFood()`. As a result, it becomes abstract. Now, all child classes of the `Animal` class would have to implement the `eatingCompleted()` method, or they will also become abstract. The entire range of animals in the zoo will have the `Animal` class as its parent for now.

🖥️ It's time to get coding!

- 🖥️ TODO 1: Create a new child class `Tiger` that extends the `Animal` class named "Tiger".
    - Create the following properties: number of stripes, speed, and sound level of roar.
    - Ensure that the variables' names represent the property they signify—for example, `numberOfStripes` for the number of stripes

value. Also, ensure that all properties are protected using `private`.
  - Create the getter and setter methods for all the properties. Remember that you can right-click on *Tiger.java* and select generate to let IntelliJ automatically generate the getter and setter methods instead of coding them manually. Ensure that the getter and setter methods are public so that the properties are accessible indirectly from outside the class.
  - Ensure that the default constructor of the `Tiger` class calls the single-argument constructor of the parent class, passing the name as “Tiger” whenever you create an object using the default constructor of `Tiger`. This guarantees that the property `nameOfAnimal` is set to the type of animal, which is a tiger in this case, as soon as it is created and does not contain the value "Unknown Animal" set by the default constructor of the `Animal` class.

```
/** TODO 1: extend the class Animal to create a new

 * land based animal named "Tiger" which

 * extends Animal.

 * Extra properties of class "Tiger" are:

 * 1. number of stripes

 * 2. speed

 * 3. sound level of roar

 *

 **/
```

- 🖥️ TODO 2: Create a new child `Dolphin` class which extends the `Animal` class named "Dolphin".
  - Create the following properties: color of dolphin and swimming speed.
  - Ensure that the variables' names represent the property they signify—for example, `swimmingSpeed`. Ensure all properties are protected using `private`.

    - Create the getter and setter methods for all the properties. Ensure getter and setter methods are public so that the properties are accessible indirectly from outside the class.
    - Ensure that the default constructor of the `Dolphin` class calls the single argument constructor of the parent class, passing the name as “Dolphin” whenever you create an object using the default constructor of `Dolphin`.

```
/** TODO 2: extend the class Animal to create a new

 * water based animal named "Dolphin" which

 * extends Animal

 * Extra properties of class "Dolphin" include:

 * 1. color of dolphin

 * 2. swimming speed

 *

 **/
```

Alright, you’ve created the `Tiger` and `Dolphin` classes by extending the `Animal` class. Now, think about how to provide specific functionalities for each animal. For the tiger, you’ll need to implement the walking functionality, and the dolphin will need a swim functionality. But both animals will have to eat, right?

Since, the `Animal` class already implements the `Eat` interface, it is automatically inherited by the child classes `Tiger` and `Dolphin`.

However, remember, the `Animal` class implements only the `eatingFood()` method and not the `eatingCompleted()` method, which has to be implemented by all child classes like `Tiger` and `Dolphin` classes to avoid becoming abstract. Why? `eatingCompleted()` needs to be unique for each animal, as tigers and dolphins finish eating in a different way. This will make your program more realistic and tailored to each animal.

🖥️ **It’s time to get coding!**

- 🖥️ **TODO 3: Create the functionality for eating which was not implemented by the parent class `Animal` for the `Tiger` and `Dolphin` classes.**

Remember

When implementing methods from an interface, always use the `@Override` annotation directly above each method definition. This annotation serves as a safety net, ensuring that you are correctly overriding the interface's methods and helping to prevent potential errors arising from typos or mismatched signatures.

- **Go to the *Tiger.java* file you created when creating the `Tiger` class in TODO 1 and implement the unimplemented methods of the `Eat` interface. Remember, you only need to implement the method `eatingCompleted()`, which will display "Tiger: I have eaten meat".**

Go to the *Dolphin.java* file you created when creating the `Dolphin` class in TODO 2 and implement unimplemented methods of the `Eat` interface.

Now suppose, you want to display a different message in the `eatingFood()` method than what is displayed in the parent class `Animal`, Then, you would need to override the first method `eatingFood()`, to display "Dolphin: I am eating delicious fish".

- **You would also need to implement the method `eatingCompleted()`, which will display "I have eaten fish", because the parent class `Animal` does not implement it.**

```
/** TODO 3: implement the unimplemented methods of

    *           "Eat" interface in the class

         *          "Tiger" class created in the TODO 1

         *          and also in the

         *          "Dolphin" class created in TODO 2.

         **/
```

Grand! Next, you will get the tiger to walk and the dolphin to swim.

🖥️ It's time to get coding!

- 🖥️ TODO 4: Create the walking functionality in the `Walk` interface in the file *Walk.java*. Any animal that walks on land can then implement this interface. In *Walk.java* and in the interface `Walk`, declare a method named `walking()`, with return type void.

Reminder

All methods declared within an interface are implicitly public, even if you don't explicitly use the `public` keyword. This means that any class implementing the interface can access and use these methods.

```
/** TODO 4: declare a functionality or method named

     *          "walking" with return type "void"

     */
```

- 🖥️ TODO 5: Make the `Tiger` class implement the walking functionality of the `Walk` interface.

```
    /** TODO 5: implement the "Walk" interface in

         *          "Tiger" class created in the TODO 1

         *          and in the  implementation of the

         *          "walking" method of the interface

         *          display -

         *          "Tiger: I am walking at the speed "

         *          and join the value of the variable "speed"

         *

         **/
```

Go to the file *Tiger.java* and implement the interface `Walk`. Now, implement the method `walking()`, which you declared in TODO 4. The method should display "I am walking at the speed", then join the value of the variable speed and "mph".

For example, if the value of the variable of speed is 30 then the display will be:

"Tiger: I am moving at the speed of 30 mph".

- 🖥️ **TODO 6: Create an interface for the swimming functionality that can be implemented by the classes of animals that swim in water. Create a new interface named `Swim` and declare a functionality or method named `swimming()` with return type void.**

```
/** TODO 6: create a new interface named "Swim"

  *         and declaring a method inside it

  *         named "swimming" with the return type

  *         "void"

  **/
```

- 🖥️ **TODO 7: Make the `Dolphin` class implement the swimming functionality of the `Swim` interface. Implement the interface `Swim` you created in TODO 6 and display the speed of the dolphin swimming in the implementation of the method `swimming()`. For example, if the speed of the dolphin is 70, it should display: "Dolphin: I am swimming at the speed of 70 nautical miles per hour".**

```
/** TODO 7: implement the "Swim" interface

  *         in the "Dolphin" class and the

  *         "swimming" method in its implementation

  *         should display the swimming speed as

  *         "Dolphin: I am swimming at the speed ...."

  *         where .... is the value of the variable

  *         "swimmingSpeed"
```

```
        **/
```

Task 3: Creating a menu system to gather information

Well done on creating animals for your virtual zoo! Next, you must utilize the menu system provided in the starter code, so users can choose between a tiger or a dolphin. Leverage the use of the provided `animalChoiceMenu()` method. The structure of the menu is provided in the *Main.java file*.

🖥️ **It's time to get coding!**

- 🖥️ **TODO 8: Utilize the menu and sub-menu to work with a particular animal selected.**

```
/** TODO 8: Utilize the menu and sub-create a menu system to

   *          work with the Animal selected

 **/
```

In order to accomplish this task, let's break it up into different steps.

TODO 8 Step 1: At the top of the `main` method, create one object of the class `Tiger` and one object of the class `Dolphin`. Name the object of the `Tiger` class `tigerObject` and the name of the object of the `Dolphin` class `dolphinObject`.

Hint

Creating the objects of the classes will enable you to call the methods of the objects when required. You can declare them at the start of the `main` method, immediately after the curly brace of the `main` method opens, so that the objects are available throughout the `main` method.

TODO 8 Step 2: A main menu system is provided to choose the animal to work with in *main.java*.

It displays:

```
******* ZOO ANIMAL choice menu ******

1. Tiger

2. Dolphin

Enter choice of animal (1-2):
```

The value you input is used in the `switch` statement of the starter code at the place which states:

```
switch (animalChoiceMenu(keyboard)) {}
```

The `switch` statement is set up to use case 1 for the object of the `Tiger` class and case 2 for the object of the `Dolphin` class.

Now, we want to display the name of the animal that was selected inside the code so that the animal chosen would appear:

```
System.out.println("The animal which is chosen is : " )
```

To do this, you would have to call the method `getNameOfAnimal()` with the appropriate object of the animal as per case 1 or case 2 being used. So, this means that if the object being selected is `Tiger`, then you would call:

```
tigerAnimal.getNameOfAnimal()
```

If you remember, you have to join this with the string "The animal which is chosen is:" using the string concatenation operator.

TODO 8 Step 3:

When a user selects an animal in the main menu, display the menu with the options to manipulate the details of the animal by leveraging the `animalDetailsManipulationMenu()` method.

Hint

You will find the line

```
System.out.println("The animal which is chosen is : ");
```

```
// get menu choice
```

in every `case` statement. You just modified the display in Step 2.

You need to call the method and assign its value to the variable `menuChoice` so that it can be used in the nested `switch(menuChoice)` statement. You can use something like this for the `Tiger` object:

```
menuChoice = animalDetailsManipulationMenu(keyboard,
tigerObject);
```

The number of `case` statements in the nested `switch` statement corresponds to the options of the menu shown by :

1. **Set properties**
2. **Display properties**
3. **Display movement**
4. **Display eating**

These four options are what you will code next.

TODO 8 Step 4:

In this step we will use the setter and getter methods of the object of the class to do what the `case` statement corresponding to the menu expects us to do.

- **Set the value of the animal characteristics. Remember that the setters are different for different animals based on their properties.**

Hint

So, for example, in case 1, you would call the method `setSpeed()` as follows, to set the speed of the tiger :

```
System.out.println("Enter speed:");

tigerObject.setSpeed(keyboard.nextInt());
```

This is just one example. You need to call all the class methods. To do this, you can go to the class definition file (for `Tiger`, you would go to *Tiger.java*, and for `Dolphin`, to *Dolphin.java*) and check the setter methods there.

- **Display the animal's characteristics. Remember that the getters you have to call are different based on the animal's properties.**

Hint

For example, in case 2, you can call the getter methods defined in the class of the object:

```
System.out.println("Age: " + dolphinObject.getAge());

System.out.println("Height: " + dolphinObject.getHeight());
```

- **Call the methods of the implemented interfaces. Remember that different animals implement different methods, so the available methods differ.**

Hint

Since the `Tiger` class implements the `Walk` interface, it has the method `walking()` whereas since the `Dolphin` class implements the `Swim` interface, it has the method `swimming()`.

- If an invalid choice is given, display “Invalid choice” for the menu option.

Pause and check

Great progress! Before you continue let's make sure everything is working so far. Run your program and observe the following:

- Remember that there are two levels of menu which will be offered to you. The first-level menu will provide you with the following options:

```
******* ZOO ANIMAL choice menu ******

1. Tiger

2. Dolphin

Enter choice of animal (1-2):
```

- When you have to choose a tiger or dolphin you have to put 1 or 2 according to the requirements. After the animal is selected, you will be provided with another menu to work with the animal chosen. Suppose you have chosen a tiger then it will show:

```
****** ANIMAL details menu for: Tiger ******

1. Set properties

2. Display properties

3. Display movement

4. Display eating

Enter choice (1-4):
```

After executing the details for the animal, you will be asked if you want to continue. This will be displayed as the following:

```
Continue with this animal? (Enter 1 for yes/ 2 for no):
```

You need to enter 1 to keep working with this animal. For example, if you are first asked to set the values of the tiger then after setting the properties, you will be asked to choose to continue. Choose 1 and then you can test the properties. If you choose 2 at this stage, you will be taken to the main menu.

Now that you have understood the menu system go ahead and test it out.

1. **Choose to view a tiger in the main menu**

Now, in the sub menu choose to do the following tasks (remember to continue with the same animal after each of the tasks until the eating behavior is done).

- **Set values for all tiger properties.**
- **Display the tiger's characteristics.**
- **Test the walking functionality.**
- **Check the eating behavior.**

2. **Choose to view a dolphin in the main menu**

Now, in the sub menu, choose to do the following tasks (remember to continue with the same animal after each of the tasks until the eating behavior is done).

- **Set the dolphin's properties.**
- **View its characteristics.**
- **Test the swimming functionality.**
- **Verify both eating methods work correctly.**

Expected output

Compare your output with the expected output.

- **For tiger, you should see its stripes, speed, and roar level, along with the common properties for an `Animal`.**
- **The tiger should walk at the speed you set.**
- **It should display "Tiger: I have eaten meat" when eating.**

- For dolphin, you should see its color and swimming speed, along with the common properties for an `Animal`.
- The dolphin should swim at the speed you set.
- It should display "Dolphin: I am eating delicious fish and I have eaten fish".

Troubleshooting

If your output doesn't match the expected output, you can try the following steps:

1. Check *Tiger.java* and *Dolphin.java* to ensure all properties are defined and have getters/setters.
2. Verify you've implemented the `Walk` interface in `Tiger` and `Swim` interface in `Dolphin`.
3. Make sure the eating methods are correctly overridden in each class.
4. In *Main.java*, confirm your menu system is calling the right methods for each animal.

Now, try to create something special - a multifunctional class for a penguin that can walk and swim! Why is this important? By designing a class that can walk and swim, you'll demonstrate the power of inheritance and interfaces to handle complex behaviors. Imagine how versatile and efficient your code will become! This task challenges you to think creatively and apply what you've learned to make your code scalable and maintainable.

🖥️ It's time to get coding!

- 🖥️ TODO 9: Create a new child class of the `Animal` class named `Penguin`.
    - Create a property `isSwimming` of boolean type to indicate whether the penguin is swimming or walking. Also, create properties `walkSpeed` and `swimSpeed`. Ensure all properties are protected using `private`.
    - Create the setters and getters of the properties. Ensure the `getter` and `setter` methods are `public` so that the properties are accessible indirectly from outside the class.

    - Ensure that when the default constructor of the `Penguin` is called, the name of the animal `Penguin` is passed to the constructor of the parent class `Animal`.
    - Override the `eatingFood()` method and implement the method `eatingCompleted()` of the `Eat` interface like the `Dolphin` class. Why? Because penguins eat fish like dolphins!
    - Implement both the `Walk` and the `Swim` interfaces.

```
/** TODO 9: create a class "Penguin" from the "Animal" class **/
```

- 🖥️ TODO 10: Modify the menu system you set up in TODO 8 to choose a penguin. Remember, if you choose a penguin, you have to ask if it is walking or swimming before you can call the methods `walking()` or `swimming()`. Why do you think this is important? Because it cannot walk and swim at the same time!
    - Create a `Penguin` object after the `Dolphin` object.
    - Modify the menu system in the method `animalChoiceMenu()` to include the option to select a penguin.
    - Add case 3 in the menu and show the same functionality for a penguin as for a tiger and dolphin.

```
                                                                                1
/** TODO 10: integrate the choice to pick a "penguin" in the menu system **/
```

You've made it to the final stretch! Now, it's time to ensure that each part of your project works together smoothly. This is your chance to test your program and see your hard work come to life! Through testing, you ensure everything runs as planned before you submit.

- Run the main program and let's get testing! Choose to view a Penguin in the main menu.

Now, in the sub menu choose to do the following tasks (remember to continue with the same animal after each of the tasks until the eating behavior is done).

- Set values for all the penguin's properties.
- Display the penguin's characteristics.

- Test the movement functionality.
- Check the eating behavior.

Expected output

Compare your output with the expected output.

- The main menu should now show an option for a penguin:

```
******* ZOO ANIMAL choice menu ******

1. Tiger

2. Dolphin

3. Penguin

Enter choice of animal (1-3):
```

- When you want to set the properties of a penguin it should ask the following questions:

```
Is the dolphin swimming (true/false):

Enter the walk speed of the penguin:

Enter the swim speed of the penguin:
```

- When you display the properties of the penguin, it should display:

```
Age:

Height:

Weight:

Walking Speed:

Swimming Speed:
```

- If the penguin's `isSwimming` property is set to true then it should display that it is swimming otherwise it should display that it is walking.
- For the eating functionality, it should display that it is eating fish.

By following these steps, you can ensure that the program works as expected for different animals and their specific characteristics. If you can complete all the outlined test steps, congratulations—your program is working well!
After you have tested and confirmed your project functionality, if you are happy with your results you can now submit your assignment for grading and feedback!

To submit your assignment, make sure to save all of your files in the IntelliJ IDE using File -> Save All, then look for and click the blue "Submit Assignment" button in the top right corner of your lab environment. This sends your code off to the grader for evaluation.

When you submit your assignment, the grader is designed to check how well your code performs its intended functions based on the instructions for this assignment. Think of it as a thorough code review!

You can submit as many times as you'd like! Use the grader as a partner to help improve your code with every submission.

Here's the crucial part: the grader needs to compile and run your application in order to work. If it can't get past this stage, your grade will unfortunately be a 0. If you submit your work and the grade is a 0, there is a good chance that your hard work couldn't compile - not necessarily that your logic or functionality is wrong. So double-check that everything compiles and runs correctly in the IDE before submitting. Please ensure the files you were working on are located in the correct `/home/coder/coursera/creating-a-zoo/src` directory as these are the files the autograder will submit, this should be the default project directory when you first opened the IDE.

Congratulations, Java maestro! You've just completed an incredible journey in creating your very own zoo program. You extended the `Animal` class to create

unique `Tiger` and `Dolphin` classes, each with distinct properties and behaviors. By implementing the `Eat`, `Walk`, and `Swim` interfaces, you ensured each animal could perform its specific actions, showcasing your understanding of inheritance and interface implementation.

You also built a user-friendly menu system, allowing users to interact with your zoo animals, and added a `Penguin` class that can both walk and swim. This demonstrated your ability to handle multiple interfaces and complex behaviors.

Your virtual zoo is now open for visitors, and you've earned your first belt in the world of Java katas. Keep this momentum going, and remember, the skills you've honed here will serve as a strong foundation for all your future programming adventures. Great work, and keep coding!