**SUTD 50.001 Introduction to Information Systems and Programming**

**Problem Set 3A**

**For all questions, please access the vocareum link found at eDimension for the starter code and to submit.**

**The Vocareum link is for submission only. Please work on the problems in Android studio, and this includes writing code for the test cases.**

**To prevent hard-coding, test cases used in Vocareum *may* be different from those provided here and will not be given to you.**

**There is only one question in this Problem Set.**

1. [48 points] We modify an earlier question in Problem Set 1B to illustrate how object creation can delegated to another class and to show how you can apply good OOP design principles to make your code flexible.

**Requirement 1.** You are given an abstract class called **Animal**. It has one abstract method **public String makeSound();**

|  |
| --- |
| public abstract class Animal {   private String name;   Animal(String name){  this.name = name;  }   public String getName(){  return name;  }   public abstract String makeSound();  } |

Add **toString()** method as a matter of good practice. Make it return the String returned by **getName().**

Add **equals()** and **hashCode()** methods as a matter of good practice. You can generate the code for this. In Android studio, just type in *equals* and select the “generate via wizard” option. Read the options carefully.

A screenshot of a computer

Description automatically generated

**equals()** is meant to return true if the contents of two objects of the same class, **this** and **other** are the same, and false otherwise. Study the generated code for **equals()** carefully. There should be code to check for:

* If **other** is not an instance of Animal, return false, otherwise   
  if the name instance variables are the same, return true, otherwise, return false
* If this and other are the same object, return true

**hashCode()** must also be overridden if **equals()** is overridden. Two objects that are equal have to return the same value for **hashCode()**. It is sufficient to use the static method Objects.hashCode() (which is generated by the Android studio code wizard).

For more details, refer to this website. <https://www.baeldung.com/java-equals-hashcode-contracts>

Create three concrete classes **Cat**, **Cow** and **Dog** that are subclasses of **Animal**. Their constructors initialize the name field as illustrated below. Override **makeSound()** in the respective classes to return a String as illustrated below.

|  |  |  |
| --- | --- | --- |
| **Constructor** | **getName() returns** | **makeSound() returns** |
| **new** Cat(“Neko”) | Cat:Neko | Cat:Neko says Meow |
| **new** Dog(“Fido”) | Dog:Fido | Dog:Fido says Woof |
| **new** Cow(“Gyudon”) | Cow:Gyudon | Cow:Gyudon says Moo Moo |

**Requirement 2.** You are given an interface **AnimalFactory** with one abstract method **createAnimal(String type, String name).**

This method is meant to take in two inputs

* **type**, which could be the Strings “Cat”, “Dog”, “Cow” or any other string
* **name,** which is a String to be passed to the constructor of **Cat()**, **Dog()** or **Cow()**

and return an Animal object. Such a design is sometimes called the “Simple Factory Pattern”.

Create a class **FarmFactory** which implements **AnimalFactory.** The following table summarizes how **createAnimal()** is to be implemented, based on the String contained in **type**.

|  |  |
| --- | --- |
| If **type** is the following String (case-sensitive): | Then this method returns the following object: |
| “Cat” | **new** Cat(**name**) |
| “Dog” | **new** Dog(**name**) |
| “Cow” | **new** Cow(**name**) |
| Any other string | null |

**Requirement 3.** You are given the class Zoo which has the following structure.

|  |
| --- |
| public class Zoo {   private AnimalFactory animalFactory;  private List<Animal> animalList;   Zoo(AnimalFactory animalFactory){  }   public void addAnimal(String type, String name){  }   public String performConcert(){  }  } |

Complete the following.

1. The **constructor** to initialize the instance variables. You are free to assign any **List<E>** concrete class to **animalList**.
2. Override **toString()** to return the **toString()** of **animalList**
3. **public void addAnimal(String type, String name)**

This invokes the **createAnimal** method on the **abstractFactory** instance variable which returns an **Animal** object based on the inputs **type** and **name**.   
  
If this **Animal** object is not **null**, it is added to **animalList**, otherwise, nothing happens.

1. **public String performConcert()**

This loops through **animalList** and returns a String which concatenates all the Strings returned by **makeSound(),** separated by an asterisk. There are no newline characters in the string returned.

Testcases are found in **TestZoo.java** and are reproduced below for your reference. TestZoo is provided in Vocareum for your convenience, and the test cases are not exhaustive. You are expected to test each class on your own.

|  |
| --- |
| public class TestZoo {  public static void main(String[] args) {   Animal cat1 = new Cat("ichika");  System.*out*.println(cat1.getName());  System.*out*.println(cat1.makeSound());  */\*\* Output:  \* Cat:ichika  \* Cat:ichika says Meow \*/* Animal cat2 = new Cat("nino");  Animal cat3 = new Cat("ichika");  System.*out*.println(cat1.equals(cat3));  System.*out*.println(cat2.equals(cat3));  */\*\* Output:  \* true  \* false \*/* Animal dog1 = new Dog("Fido");  System.*out*.println(dog1.getName());  System.*out*.println(dog1.makeSound());  */\*\* Output  \* Dog:Fido  \* Dog:Fido says Woof  \*/* AnimalFactory animalFactory = new FarmFactory();  Animal animal1   = animalFactory.createAnimal("Cat","miku");  Animal animal2   = animalFactory.createAnimal("Dog","hachiko");  System.*out*.println(animal1.getName());  System.*out*.println(animal2.getName());  */\*\* Output  \* Cat:miku  \* Dog:hachiko \*/*  Zoo zoo = new Zoo( new FarmFactory());  System.*out*.println(zoo);  zoo.addAnimal("Cat", "Kuroneko");  zoo.addAnimal("Dog", "Bond");  zoo.addAnimal("Cow", "Gyudon");  System.*out*.println(zoo.performConcert());  System.*out*.println(zoo);  */\*\* Output*  *\* []  \* Cat:Kuroneko says Meow\*Dog:Bond says Woof\*Cow:Gyudon says Moo Moo\*  \* [Cat:Kuroneko, Dog:Bond, Cow:Gyudon]  \*/*  } |