Code Challenge:

Let's do some code practice. Use Intellij IDEA for this Code Challenge.

Imagine a Cake. When we think of a Cake we usually think of its finger licking flavour and while buying a Cake we first check its availability along with its price in the bakery shop. So let us create a class for Cake. The class Cake should contain instance variable of cakeName, price, flavour and isAvailable or not.

After you have created a class of Cake, create an object for your favorite cake. Mine is "Black Forest", so I will go ahead and create its object. You too do the same.

//Cake.java

package com.internshala.javaapp;

public class Cake {

public String cakeName;

public String flavour;

public float price;

public boolean isAvailable;

}

//Main.java

package com.internshala.javaapp;

public class Main {

public static void main(String[] args) {

Cake cake = new Cake();

cake.cakeName = "Black Forest";

cake.flavour = "Chocolate";

cake.price = 280.0f;

cake.isAvailable = true;

}

}

Q. Why are getters and setters used?

Getter and setter are generally used with private methods. These are used to ensure abstraction. Since private members cannot be accessed from outside the scopeof the class so to get their values and to set their values we make public getter and setter methods.

Code Challenge:

Inside the Intellij IDEA modify the code within Main.java to implement Multi-level Inheritance. Create a new class Puppy that contains instance variables of "size" and "age". The Puppy should inherit from class Dog which in turn inherits from class Animal. Finally, test your code by accessing properties and behavior of super class Dog and Animal using Puppy object.

package com.internshala.javaapp;

public class Main {

public static void main(String[] args) {

Puppy puppy = new Puppy();

puppy.name = "Bruno";

puppy.color = "Brown";

puppy.age = 7;

puppy.size = 40;

puppy.bark();

puppy.run();

// Yeah we are able to access Dog and Animal class properties via Puppy object.

}

}

class Animal {

String name;

public void run() {

System.out.println("Animal is running !");

}

}

class Dog extends Animal {

String color;

public void bark() {

System.out.println("Wooh ! Wooh !");

}

}

class Puppy extends Dog {

int size;

int age;

}

Method overriding is \_\_\_\_\_\_\_\_\_\_\_\_

* Compile Time Polymorphism
* Runtime Polymorphism
* Runnable Polymorphism

Example of Compile time polymorphism is \_\_\_\_\_\_\_\_\_\_\_\_\_

* Method Overriding
* Constructor overloading
* Method overloading
* Both Constructor and Method overloading

In order to call Parent class constructor or method we use \_\_\_\_\_\_\_\_\_\_\_ keyword.

* super
* this
* parent
* abstract

The polymorphism example where we have same method name but different parameters is \_\_\_\_\_\_\_\_\_\_\_

* Inheritance
* Abstraction
* Overloading
* Overriding

The polymorphism example where the methods have same signature is \_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Overloading
* Overriding //answer
* Abstraction
* Inheritance

abstract methods does not contain \_\_\_\_\_

* method body
* parameters
* return type
* modifier

You cannot instantiate or create objects of abstract class.

* False
* True
* Varies from case to case
* || TABLE

1) The abstract keyword is used to declare abstract class. |||| The interface keyword is used to declare interface.

2) We use extends keyword for inheritance |||| We use implements keyword for implementing interface in a class.

3) Abstract class can have abstract and non-abstract methods. |||| Interface can have only abstract methods.

4) Abstract class doesn't support multiple inheritance. |||| Interface supports multiple inheritance.

5) Abstract class can have final, non-final, static and non-static variables. |||| Interface has only static and final variables.

The methods of the interface are by default \_\_\_\_\_\_

* only abstract
* abstract and public
* only public
* static and public

To declare a constant value, the appropriate modifier is \_\_\_\_\_\_\_\_\_\_

* static final
* final
* private static
* public static

Static variables are also known as \_\_\_\_\_\_\_\_\_\_\_\_

* Local variables
* Instance variables
* Class variables
* Objects

Static methods can access \_\_\_\_\_\_\_\_\_\_\_

* Only static variables
* Both static variables and static methods
* Only static methods
* static variables, static methods and instance variables
* Apart from what we learned in the video there are few other commonly used methods to perform operations on the ArrayList. They are:
* 1. void add(int index, Object element): It is used to insert the specified element at the specified position index in a list.
* 2. remove(int index): Removes the element at the specified position in the list.
* 3. int size(): Returns the size of the list.
* 4. void clear(): It is used to remove all of the elements from this list.
* 5. int lastIndexOf(Object o): It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
* 6. Object[] toArray(): It is used to return an array containing all of the elements in this list in the correct order.
* 7. Object[] toArray(Object[] a): It is used to return an array containing all of the elements in this list in the correct order.
* 8. boolean add(Object o): It is used to append the specified element to the end of a list.
* 9. int indexOf(Object o): It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
* Apart from what we learned in the video, there are many other methods to perform operations on HashMap. They are:
* 1. void clear(): It is used to remove all of the mappings from this map.
* 2. boolean containsKey(Object key): It is used to return true if this map contains a mapping for the specified key.
* 3. boolean containsValue(Object value): It is used to return true if this map maps one or more keys to the specified value.
* 4. boolean isEmpty(): It is used to return true if this map contains no key-value mappings.
* 5. Set entrySet(): It is used to return a collection view of the mappings contained in this map.
* 6. Set keySet(): It is used to return a set view of the keys contained in this map.
* 7. int size(): It is used to return the number of key-value mappings in this map.
* Apart from List and Map there are various other interface that are a part of Java Collection Framework:
* 1. Set — It is very similar to List interface but a Set cannot contain duplicate elements whereas a List can contain duplicate entries.
* 2. Queue — It is a collection that follows a principle of FIFO i.e. First In First Out. According to FIFO, the element that enters the Queue first will be removed from Queue first. It can be compared to a queue in front of a ticket counter where each person is served on the basis of first come and first basis i.e. a person who joins the queue first leaves the queue first after getting served.
* 3. Deque — Deques can be used both as FIFO (first-in, first-out) and LIFO (last-in, first-out). In a deque all new elements can be inserted, retrieved and removed at both ends.
* Q. Where do we need to use ArrayList and HashMap?
* ArrayLists and HashMaps are different data structures. HashMaps are used for when you want to associate a key with a value and ArrayLists are an ordered collection.
* Q. What is static keyword?
* Static members belong to the class instead of a specific instance.
* It means that only one instance of a static field exists even if you create a million instances of the class or you don't create any. It will be shared by all instances.
* Since static methods also do not belong to a specific instance, they can't refer to instance members (how would you know which instance Hello class you want to refer to?). Static members can only refer to static members. Instance members can, of course access static members.
* Static members can also access instance members through an object reference.
* Q. What is' implements' in java?
* 'implements' keyword comes to use when we deal wit the interface. An interface is a
* reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements
* an interface, thereby inheriting the abstract methods of the interface.
* Q. What is the need for abstract classes if you can define each function seperately in as many  
  functions as you like with the same name even without creating an abstract class
* The abstract class helps us define a template for children classes. e.g. Animal abstract class
* can have abstract methods like movement() and talkingStyle() which is common for all animals so we
* can have a well-structured program. Abstract Classes are a good fit if you want to provide
* implementation details to your children but don't want to allow an instance of your class to be
* directly instantiated (which allows you to partially define a class). Yes you can achieve the same
* without the use of abstract class but instead of creating the same method for different classes, you can
* have one central superclass from which other classes inherit and just implement their own version
* of that method.
* [see more](https://trainings.internshala.com/content/java)
* Q. Why are getters and setters used?
* Getter and setter are generally used with private methods. These are used to ensure abstraction. Since private members cannot be accessed from outside the scopeof the class so to get their values and to set their values we make public getter and setter methods.

The class used to open a existing file and write data to it is \_\_\_\_\_\_\_\_\_\_\_\_

* FileInputStream
* FileOutputStream
* FileIOStream
* All of these

The end of file is marked by \_\_\_\_\_\_\_\_\_\_

* 1
* 0
* -1
* null

The data type used to write data to a file is \_\_\_\_\_\_\_\_\_\_\_\_

* byte array
* int array
* String
* Any of these

Why it is important to close the FileInputStream and FileOutputStream after we are done processing the file?

* To make sure data is securely saved in the file
* To avoid resource or memory leakage
* To avoid same data being written to the file twice
* All of these mentioned are correct
* Object Oriented Programming
* >
* Module Test Summary
* Object Oriented Programming Test Summary
* Your Marks 58%
* 1 Which of these keywords is used to make a class?
* Attempted answer: None of the mentioned
* Correct answer: class
* 2 The class used to open a existing file and read from it is \_\_\_\_\_\_\_\_
* Attempted answer: FileInputStream
* 3 A \_\_\_\_\_\_\_\_\_ is a template or a blue print from which objects are created.
* Attempted answer: class
* 4 A variable which is present within a class but outside the methods is called \_\_\_\_\_\_\_\_\_\_\_
* Attempted answer: Instance variable
* 5 Which keyword is used to call parent class constuctor and method from wihin the child class?
* Attempted answer: super
* 6 Find the output:  
    
  class Main {     public static void main(String args[]) {  
      Car car = new Car();  
      car.increment();  
      System.out.println(car.x);  
  }  
  }  
    
     class Car {  
     int x = 0;  
      void increment(){  
     x++;  
     x++;  
    }  
  }
* Attempted answer: 0
* Correct answer: 2
* 7 Consider the following code snippet:
* interface ClickEvent {  
  // Your code...  
  }
* class Button implements ClickEvent {  
  // Your code...  
  }
* Which of the following is valid?
* Attempted answer: ClickEvent ref = new ClickEvent();
* Correct answer: ClickEvent ref = new Button();
* 8 Which of these keyword must be used to inherit a class?
* Attempted answer: super
* Correct answer: extends
* 9 Find the output:
* class Calculate {  
    
  int x;  
  int y;  
    
  void add(int a) {  
  x = a + 1;  
  }
* void add(int a, int b){  
  x = a + 2;  
  }   
  }
* class Main {  
  public static void main(String args[])  
  {  
  Calculate obj = new Calculate();   
  obj.add(6);  
  System.out.println(obj.x);   
  }  
  }
* Attempted answer: 7
* 10 What is the process of defining a method in subclass having same name & type signature as a method in its superclass?
* Attempted answer: Method overriding