**1. Study following object-oriented concepts**

1. Object: Simply put, if we look around in the real world, we will find dozens or even hundreds of objects around us. We have always been surrounded by objects, my laptop is an object, the phone is regarded as an object, a table, a chair, a table, a mouse, a glass of water, and even I am regarded as an object, all of which are considered objects in the real world of objects, And it can be expressed and converted into code using an object-oriented programming language such as Java. Objects are the core and most basic element of Java, so if we implement real-world objects in Java code, it is easy to do this.

2. Class: If we have dozens or hundreds of different and random objects, we want to divide them into groups or "classify" them based on their common characteristics. The word "class" comes from "classification" because we can classify objects with similar characteristics and characteristics as the parent class. For example, if we have real-world objects such as BMW, Mercedes-Benz and iPhone then we will group or classify these objects based on their common characteristics. Therefore, BMW and Mercedes will be the objects of the "car" category, while the iPhone will be the objects of the parent "phone".

3. Instantiation of an object: Instantiation in object-oriented programming means creating an object that inherits all the parental characteristics of the class. Therefore, upon instantiation, this means that you are creating a new object that will inherit or save the properties and characteristics of the parent class from which it was instantiated. For example, if we have an Employee class with attributes or characteristics such as name, department, and job title, then if we instantiate an object in this class, we will have an employee named Adnan Naseer, working in the IT support department , And software developers as posts.4. Visibility (private / protected / public): Private, public and protected people can access these projects. For example, if the object is set to private, it means that the object cannot be accessed or used outside its class; if the object is set to protected, it means that the object cannot be used outside its package (which means it Can be used by subclasses or subclasses), if you set it to public, you can access it anywhere, in all different classes and even packages. The same idea applies to methods and classes.

5. Member data / methods: The method is the same as the function (known in JavaScript). The method is a block of code that is executed only when called. You can think of methods as the behavior of objects. For example, if we have a dog as an object, then the dog's method (or function) will be barking or eating. This real-world feature or behavior can be converted into code blocks, which is exactly the function of the method or function.

6. Inheritance: The meaning or function can be clearly seen from its name. Inheritance is a broad term and can also be applied to classes and objects. Inheritance refers to when a subclass (sometimes called a subclass) inherits the properties and methods of its parent class (superclass). This means that the subclass can use the methods and objects declared in its parent class, because the subclass is derived from the parent class.

7. Interface: Interface is another attribute in object-oriented programming. It is considered higher than the class in the hierarchy, which means that the class is derived from the interface. The interface is similar to the function of the class, similar to the call and structure of the class. For a simpler explanation, this interface is used to set an abstract method by default (meaning it has no body, and is not set to do nothing by default), so that all the classes created in the interface are shared These methods. Inherit this interface. For example, if we have three categories, namely jet, car and motorcycle, then these three categories share a common function, that is to start the engine. Therefore, we want to set up a method in which all three classes have the same method, but abstract this method (without the body) because all three classes have different ways to start the engine. Therefore, the body of the "start engine" method of each class can be different. In this case, we use a declarative interface in which to create this function (method), which will be used in all inherited classes, and then these three classes can use this method and modify its body (actual function code ) To suit its needs.

8. Polymorphism: "Polymorphism" literally means "having many different forms", and this is exactly what it means. In a real-world example, polymorphism will be applied to people with different characteristics, for example, women who are also wives, mothers, and company employees. In terms of object-oriented programming, this means that a function (method) can be used multiple times, even with the same name, but with different actions and results. This can be done by making the two methods have the same name (so they are the same method) but passing two different parameters to them. For example, we can use a method called "addNumbers" that has the function of adding two numbers, but one method doubles the number in the parameter and the other uses the number as an integer in the parameter. Even, we can have a method called twice, which takes the same type of parameters in the parameters, such as integers, and the first method takes two numbers as parameters, and four numbers as parameters, and finally Two separate results are obtained for two methods with the same name. Another basic example of polymorphism in Java is that the same method declared in a superclass can be called in all subclasses, and all subclasses have different outputs. Therefore, you only have one method called once, but the method provides different output on each subclass.

9. Overriding: Overriding also means over-writing, which means literally replacing code above another code that has been declared. When the method has "@Override" at the top, it means that the method exists in the parent class and you are now overriding this method in the child class

10. Abstract classes: Abstract classes are classes that cannot be instantiated, which means that you cannot create a new object that inherits its properties, but you can inherit them as a subclass (which means you can still create a subclass that inherits its properties). . Derived from abstract classes are abstract methods that have no subjects. The methods of these subjects are created in subclasses that inherit it. Although this does not mean that all methods in the abstract class must also be abstract, non-abstract methods can be created in the abstract class.

Here is the implementation of the above defined terms in code.

1. Object:

package com.example.adnan\_1;  
  
//In order to create a an object, we must create a class named "Car"  
public class Car {  
  
//Every class should have at least one method, therefore here we create a method   
//in order to correctly create an object  
 public static void main(String[] args){  
   
//Here we create a new object named "BMW" that derives from the class "Car"  
 Car BMW = new Car();  
 }  
}

1. Class:

package com.example.adnan\_1;  
  
//Using the same example, creating a class is as simple as using the keyword //'class', followed by the name of the class that you would like to assign to //it. Of course, we also have to add a prefix keyword to indicate whether this //class is public, private, or protected  
public class Car {  
  
}

1. Instantiation of an object:

package com.example.adnan\_1;  
  
//Here we first created a class named "Car" in which we will create an object //in  
public class Car {  
  
//this is a private (accessible inside the class) method declared inside of //the parent class of the object  
 private void welcome(){  
 System.*out*.println("Welcome to the car industry");  
 }  
  
 public static void main(String [] args){  
  
//Here we instantiate an object, meaning that we create or assign a new object //that inherits the properties of its parent class, which in this case is the //"Car" class  
 Car BMW = new Car();  
  
//Here is an example of the new object we created with a method attached to it //(called on it), this is the same method that was created in the parent class //of this object. Therefore, the object //was able to inherit the "welcome"  
//function from its parents class when we instantiated it  
 BMW.welcome();  
 }  
}

1. Visibility:

package com.example.adnan\_1;  
  
public class Car {

//Here we created a private method which cannot be called or used outside of //this class  
 private void welcome(){  
 }  
  
//Here we created a public method which can be accessed and used anywhere in //the app  
 public void startEngine(){  
 }  
  
//Here we created a protected method which can be used outside of this class, //but not outside of this package (it has more access than a private method)  
 protected void stopEngine(){  
 }  
  
 public static void main(String [] args){  
 //Instantiating an object (creating a new object)  
 Car BMW = new Car();  
  
//All the methods can be called and accessed here since the object is still //inside the same class  
 BMW.welcome();  
 BMW.startEngine();  
 BMW.stopEngine();  
 }  
  
}

1. Methods:

package com.example.adnan\_1;  
  
public class Car {  
  
//This is a method, also known as a function, has its name in yellow.

//The prefixes are reserved keywords in Java that indicate the method's //behavior (in this case it is public and void, which means it doesn't return //anything)  
 public void welcome(){  
  
//Here comes the body of the method, which is the actual code being run (the //function's action)  
//This simple method prints "Hello World" on the console log.  
 System.*out*.println("Hello World");  
 }  
}

1. Inheritance:

package com.example.adnan\_1;  
  
  
//Here we create a class named "SportsCar" which inherits all the methods and //properties of its parent class (super-class) "Car"  
//Inheritance is done using the keyword "extends" in which it always the child //class that extends the parent class because the subclass is inheriting all //the properties of the super-class  
public class SportsCar extends Car {  
}

1. Interface:

package com.example.adnan\_1;  
  
//Here we declare an interface, in the same way we declare a class in  
interface Car{  
  
//In interfaces, the methods are abstract by default, which means that the //bodies of the methods are kept empty  
 public void startEngine();  
 public void stopEngine();  
}  
  
//Here we create a class which will inherit the interface's properties (the //two methods)  
//It is very similar to how we declare a child class, just instead of //"extends" we use" inherits when inheriting from interfaces  
class ElectricCar implements Car{  
  
 //Here we call the two abstract methods from the interface  
 public void startEngine() {  
 //Now we can declare the method's body  
 System.*out*.println("Press the Start Button");  
 }  
  
 public void stopEngine() {  
 System.*out*.println("Press the Stop Button");  
 }  
}

1. Polymorphism:

package com.example.adnan\_1;  
  
//This is the parent class (super-class) which we named 'Animal'  
class Animal{  
  
//In this parent class, we declare a public method that can be used in all //child or sub-classes that extend from this class  
 public void animalSound(){  
 System.*out*.println("animal sound");  
 }  
}  
  
//Here we created a child class that extends the super-class 'Animal' //(inherits all the traits and properties of the parent class)  
class Dog extends Animal{  
  
//Here is what polymorphism looks like:  
//We can re-use the same method (with the same name) as declared in the super //class, yet what makes it different is the parameter we enter into it, which //will therefore give us a different output  
 public void animalSound(){  
 System.*out*.println("woof woof");  
 }  
}  
  
//This is another child class that derives from the super class 'Animal'  
class Cat extends Animal{  
  
//Once again, we use the same method for the third time, with the only //difference being the change of the parameter that is passed into it, which //in this case is the string 'meow'. Therefore, this outcome will be different //from the Dog class and even from the super class's method  
 public void animalSound(){  
 System.*out*.println("meow");  
 }  
}

1. Overriding:

package com.example.adnan\_1;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
  
//Here, the keyword 'Override' is used on a protected method called 'onCreate'  
/This method is the default method will be executed once the app runs, //therefore it must override (over-write) any function with the exact same //name, same parameters, and same output.  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 }  
}

1. Abstract Class:

package com.example.adnan\_1;  
  
//Here we declare an abstract class, by simply adding the keyword 'abstract' //in front of the class declaration  
abstract class Car{  
  
 //Abstract classes can have both abstract and non-abstract methods  
 //Here is an example of an abstract method, therefore we have not declared  
 //its body  
 public abstract void startEngine();  
  
//At the same time, we can have a non-abstract method inside the abstract //class, just like this one:  
 public void increaseSpeed(){  
 System.*out*.println("Press the gas pedal to increase speed of car");  
 }  
  
}

**1. Study Android fundamental concepts**

1. **What programming languages you can use for Android app development?**

Android apps can be written in Java, Kotlin and C++.

1. **What is .apk file?**

APK stands for Android Package Kit, it is an archive package file that can manage all the applications released and installed by Android. Like Windows has an .exe file, it is an important file at the core of the android operating system.

1. **How Android system runs apps?**

The Android operating system is derived from the Linux operating system; therefore, it runs on the Linux kernel. The Android operating system assigns unique IDs to mobile applications in order to organize and process them separately, so that each application on the mobile application is processed differently. The unique ID assigned to the application can help the Android system to set a limited permission for each application based on its needs as to whether it can be accessed from the mobile phone system. In addition, unless the application is granted to the application, it cannot directly access all functions of the application. An application consists of components, which are the core building blocks of any android application, and the Android system will only run the components that need to run, and will not run any unused or unnecessary components in the application. The applications are isolated in the Android operating system and run using VMs or virtual machines to isolate other applications from currently running applications.

1. **Name four types of Android components. Describe each.**
2. Activities: The active component is the content that the user interacts with and experiences when using the application, which is basically the layout or screen of the application that the user sees. The active components are independent of each other, which means that each individual screen or user interface is an independent screen. The advantage of this is that if any other application needs to use a specific screen of an application, it can be easily achieved without rendering all the screens of the application. For example, if an application needs to request a contact list for your phone, it can only present a screen or user interface with a list of names and numbers. The activity is also mainly used to track what the user sees (which screen he is currently on), track which previous screens have any processes they are running on (to keep it running, assume that the user may want to return to it) Kill unused processes so that users can return to the last screen they used (with save / store status) and implement a simple user-friendly flow between the various screens of the application.
3. Services: In general, service components are components that are responsible for running processes and services in the background. However, there are two main types of service components, one is the "start service" component, and the other is the "bound service" component. Starting a service usually runs the process temporarily in the background until a process is completely killed before it completes its task. There are two forms of starting the service: one deals with processes such as music playback, where the application strictly tells the android system to let the process run for the user (because this is critical to user satisfaction), and the other deals with The normal process. The background service runs without user notification or even without the user's awareness. The regular background services of these running processes that need to be completed may spend some time at runtime, or even be killed and restarted, because they will not affect the user's interface experience. As for the second main type of service, binding service components, these services ensure that the process that another application depends on remains running and will not be killed or spend a lot of time because they are critical to the user interface. An application may be using the API and always need to get new data or synchronize new data into it, which is the task of the Bound Services component. The binding service considers the auxiliary background process that is essential to the application currently used by the user as an important factor, so it is forced to continue running like the music playback in the startup service. Examples of binding services include live wallpapers, application notifications, etc.
4. Broadcast Receivers: The broadcast receiver is a component whose main task is to assume the process of an application that does not need to run in the background before the process is completed. For example, an application that is used as a timer, if the user sets a reminder 2 weeks in advance on a specific date, the application will send this process to the broadcast receiver component to handle the operation of the process, rather than let the application Run in the background of the mobile device for 2 weeks. The broadcast receiver allows the application's running tasks to communicate with the device's system and perform some of its processes and tasks. Similarly, in some cases, the broadcast receiver component also allows the system to talk to the application. For example, if the device's battery is low, the system can send a notification to the application to remind the device that the battery has been turned low. This example can be seen in some gaming applications, which will warn the user that their battery is low and the game will pause until the user plugs into their device. For example, this process is done through the broadcast receiver component. The broadcast receiver can also send notifications to applications that are not currently running.
5. Content providers: The content provider component manages multiple applications that may require or share data from a specific application to a secure storage location; it can be a file in the system, a database such as SQLite on the Web, or an application can access it from Any other storage location for data. Each application is independent of other applications and has this location for storing files or accessing shared data. The main advantage of this content provider is that there are many applications that may eventually request the same data, such as the device ’s image (photo library), current time, or contact information. Separating all of this information can be useful for any All applications of the application, all files are managed by one file, the necessary data needs to be retrieved from it. All these data items are provided with a URI (Uniform Resource Identifier), which is actually a pointer to a specific location of the data item. The ability of each data segment to have a URI is critical to security, because the provider of the data segment can restrict which applications can use the data segment, and if they do so, they can only obtain temporary URI permissions on the data segment Access data items, not the entire data set. The data set of the device's photo library and contact information is a content provider component in the android device system.
6. **What is manifest file and what is its purpose?**

The AndroidManifest file is an XML file. Before running any component in the application, the Android system opens and scans the file to check whether the component exists in the application. The main function of this file is to summarize and save all the components included in the application and declare all the hardware and software that the application will use, such as the device's photo library, camera, contact list and information. In addition, the manifest file contains information about the API level being used (API level is a number indicating the version of the API framework used by a specific Android platform version), as well as any third-party or external libraries used by the application, such as Google Maps. The manifest file is an important part of the application. It can be run by the Android system because it mainly contains all the necessary information about the four components of activities, services, container providers, and broadcast providers, and how they interact. . All permissions of the application are also included in the manifest file.

1. **What are resources? Why they are needed?**

Resources are all files that you add to an application except code files. Resources include the main parts of the application, such as images, specific style layouts, colors, menus, animations, navigation bars, and so on. For example, your app ’s logo is a resource (picture resource), and you need to apply it to the app ’s user interface layout. Resources are a practical and useful tool for organizing all the necessary items in my application without having to explicitly write them in my code. All resources in the application have unique IDs, and you can use the unique IDs to call resources wherever needed. The advantage of using these resources without explicitly writing them to code is that you can manage the flexibility of application design. For example, you can set up a resource for users to use the application on a wide range of devices such as tablets and set up other resources for users who use smartphones. In both cases, your application still looks good. These resources also allow you to set up multiple XML files using the translated text, which will be used as the layout displayed to the user when the user selects another language.