ASSIGNMENT-6

1)OOPS Concepts

OOP concepts in Java are the main ideas behind Java’s Object Oriented Programming. They are an abstraction, encapsulation, inheritance, and polymorphism. Grasping them is key to understanding how Java works. Basically, Java OOP concepts let us create working methods and variables, then re-use all or part of them without compromising security.

There are four main OOP concepts in Java. These are:

Abstraction. Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don’t need to know how it works in order to enjoy it. In Java, abstraction means simple things like objects, classes, and variables represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.

Encapsulation. This is the practice of keeping fields within a class private, then providing access to them via public methods. It’s a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.

Inheritance. This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.

Polymorphism. This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is method overloading. That’s when different meanings are implied by the code itself. The other form is method overriding.

How Abstraction Works

Abstraction as an OOP concept in Java works by letting programmers create useful, reusable tools. For example, a programmer can create several different types of objects. These can be variables, functions, or data structures. Programmers can also create different classes of objects. These are ways to define the objects.

For instance, a class of variable might be an address. The class might specify that each address object shall have a name, street, city, and zip code. The objects, in this case, might be employee addresses, customer addresses, or supplier addresses.

How Encapsulation Works

Encapsulation lets us re-use functionality without jeopardizing security. It’s a powerful OOP concept in Java because it helps us save a lot of time. For example, we may create a piece of code that calls specific data from a database. It may be useful to reuse that code with other databases or processes. Encapsulation lets us do that while keeping our original data private. It also lets us alter our original code without breaking it for others who have adopted it in the meantime.

How Inheritance Works

Inheritance is another labor-saving Java OOP concept. It works by letting a new class adopt the properties of another. We call the inheriting class a subclass or a child class. The original class is often called the parent. We use the keyword extends to define a new class that inherits properties from an old class.

How Polymorphism Works

Polymorphism in Java works by using a reference to a parent class to affect an object in the child class. We might create a class called “horse” by extending the “animal” class. That class might also implement the “professional racing” class. The “horse” class is “polymorphic,” since it inherits attributes of both the “animal” and “professional racing” class.

Two more examples of polymorphism in Java are method overriding and method overloading.

In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That allows a programmer to use one method in different ways depending on whether it’s invoked by an object of the parent class or an object of the child class.

In method overloading, a single method may perform different functions depending on the context in which it’s called. That is, a single method name might work in different ways depending on what arguments are passed to it.

2)What is RecyclerView and how it is used?

The RecyclerView widget is a more advanced and flexible version of ListView.

In the RecyclerView model, several different components work together to display your data. The overall container for your user interface is a RecyclerView object that you add to your layout. The RecyclerView fills itself with views provided by a layout manager that you provide. You can use one of our standard layout managers (such as LinearLayoutManager or GridLayoutManager), or implement your own.

The views in the list are represented by view holder objects. These objects are instances of a class you define by extending RecyclerView.ViewHolder. Each view holder is in charge of displaying a single item with a view. For example, if your list shows music collection, each view holder might represent a single album. The RecyclerView creates only as many view holders as are needed to display the on-screen portion of the dynamic content, plus a few extra. As the user scrolls through the list, the RecyclerView takes the off-screen views and rebinds them to the data which is scrolling onto the screen.

The view holder objects are managed by an adapter, which you create by extending RecyclerView.Adapter. The adapter creates view holders as needed. The adapter also binds the view holders to their data. It does this by assigning the view holder to a position, and calling the adapter's onBindViewHolder() method. That method uses the view holder's position to determine what the contents should be, based on its list position.

This RecyclerView model does a lot of optimization work so you don't have to:

When the list is first populated, it creates and binds some view holders on either side of the list. For example, if the view is displaying list positions 0 through 9, the RecyclerView creates and binds those view holders, and might also create and bind the view holder for position 10. That way, if the user scrolls the list, the next element is ready to display.

As the user scrolls the list, the RecyclerView creates new view holders as necessary. It also saves the view holders which have scrolled off-screen, so they can be reused. If the user switches the direction they were scrolling, the view holders which were scrolled off the screen can be brought right back. On the other hand, if the user keeps scrolling in the same direction, the view holders which have been off-screen the longest can be re-bound to new data. The view holder does not need to be created or have its view inflated; instead, the app just updates the view's contents to match the new item it was bound to.

When the displayed items change, you can notify the adapter by calling an appropriate RecyclerView.Adapter.notify…() method. The adapter's built-in code then rebinds just the affected items.

Add the support library

To access the RecyclerView widget, you need to add the v7 Support Libraries to your project as follows:

Open the build.gradle file for your app module.

Add the support library to the dependencies section.

3)What is CardView and how it is used?

Apps often need to display data in similarly styled containers. These containers are often used in lists to hold each item's information. The system provides the CardView API as an easy way for you show information inside cards that have a consistent look across the platform. These cards have a default elevation above their containing view group, so the system draws shadows below them. Cards provide an easy way to contain a group of views while providing a consistent style for the container.

In order to use the CardView you need to add it to your layout file. Use it as a view group to contain other views.

The cards are drawn to the screen with a default elevation, which causes the system to draw a shadow underneath them. You can provide a custom elevation for a card with the card\_view:cardElevation attribute. This will draw a more pronounced shadow with a larger elevation, and a lower elevation will result in a lighter shadow. CardView uses real elevation and dynamic shadows on Android 5.0 (API level 21) and above and falls back to a programmatic shadow implementation on earlier versions.

Use these properties to customize the appearance of the CardView widget:

To set the corner radius in your layouts, use the card\_view:cardCornerRadius attribute.

To set the corner radius in your code, use the CardView.setRadius method.

To set the background color of a card, use the card\_view:cardBackgroundColor attribute.

3)What is an adapter and how it is used?

To feed all your data to the list, you must extend the RecyclerView.Adapter class. This object creates views for items, and replaces the content of some of the views with new data items when the original item is no longer visible.

The layout manager calls the adapter's onCreateViewHolder() method. That method needs to construct a RecyclerView.ViewHolder and set the view it uses to display its contents. The type of the ViewHolder must match the type declared in the Adapter class signature. Typically, it would set the view by inflating an XML layout file. Because the view holder is not yet assigned to any particular data, the method does not actually set the view's contents.

The layout manager then binds the view holder to its data. It does this by calling the adapter's onBindViewHolder() method, and passing the view holder's position in the RecyclerView. The onBindViewHolder() method needs to fetch the appropriate data, and use it to fill in the view holder's layout. For example, if the RecyclerView is displaying a list of names, the method might find the appropriate name in the list, and fill in the view holder's TextView widget.

If the list needs an update, call a notification method on the RecyclerView.Adapter object, such as notifyItemChanged(). The layout manager then rebinds any affected view holders, allowing their data to be updated.

4)What is TypeCasting?

Type casting refers to changing an variable of one data type into another. The compiler will automatically change one type of data into another if it makes sense. For instance, if you assign an integer value to a floating-point variable, the compiler will convert the int to a float. Casting allows you to make this type conversion explicit, or to force it when it wouldn’t normally happen.

Type conversion in c can be classified into the following two types:

**1. Implicit Type Conversion**

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as implicit type conversion or type promotion.

int x;

for(x=97; x<=122; x++)

{

printf("%c", x); /\*Implicit casting from int to char thanks to %c\*/

}

**2. Explicit Type Conversion**

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting.

Type casting in c is done in the following form:

(data\_type)expression;

where, data\_type is any valid c data type, and expression may be constant, variable or expression.

For example,

int x;

for(x=97; x<=122; x++)

{

printf("%c", (char)x); /\*Explicit casting from int to char\*/

}

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.
3. All character types to be converted to integer.

**Example**

Consider the following code:

int x=7, y=5 ;

float z;

z=x/y; /\*Here the value of z is 1\*/

If we want to get the exact value of 7/5 then we need explicit casting from int to float:

int x=7, y=5;

float z;

z = (float)x/(float)y; /\*Here the value of z is 1.4\*/

5)What is adapter in Android Studio?

An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set.

In Android, Adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to an Adapter view then view can takes the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc. For more customization in Views we uses the base adapter or custom adapters.

To fill data in a list or a grid we need to implement Adapter. Adapters acts like a bridge between UI component and data source. Here data source is the source from where we get the data and UI components are list or grid items in which we want to display that data.