**Assignment 5**

Object Oriented Programming (Oops)

A Java class uses variables to define data fields and methods to define actions. Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

There are three main features of OOPS.

1) Encapsulation

2) Inheritance

3) Polymorphism

**Encapsulation**

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a “black box,” or a device that you can use without regard to the internal mechanisms. A programmer can access and use the methods and data contained in the black box but cannot change them.

**Inheritance**

An important feature of object-oriented programs is inheritance—the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability.

**Polymorphism**

In Core, Java Polymorphism is one of easy concept to understand. Polymorphism definition is that Poly means many and morphos means forms. It describes the feature of languages that allows the same word or symbol to be interpreted correctly in different situations based on the context.

**Static Polymorphism**

The ability to execute different method implementations by altering the argument used with the method name is known as method overloading.

**Dynamic Polymorphism**

When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original superclass.

Handler

A Handler allows you to send and process [Message](https://developer.android.com/reference/android/os/Message.html) and Runnable objects associated with a thread's [Message Queue](https://developer.android.com/reference/android/os/MessageQueue.html). Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

**Event handlers**

If you're building a custom component from View, then you'll be able to define several all back methods used as default event handlers. In the document about [Custom View Components](https://developer.android.com/guide/topics/ui/custom-components.html), you'll learn some of the common callbacks used for event handling, including:

* [onKeyDown(int, KeyEvent)](https://developer.android.com/reference/android/view/View.html#onKeyDown(int,%20android.view.KeyEvent)) - Called when a new key event occurs.
* [onKeyUp(int, KeyEvent)](https://developer.android.com/reference/android/view/View.html#onKeyUp(int,%20android.view.KeyEvent)) - Called when a key up event occurs.
* [onTrackballEvent(MotionEvent)](https://developer.android.com/reference/android/view/View.html#onTrackballEvent(android.view.MotionEvent)) - Called when a trackball motion event occurs.
* [onTouchEvent(MotionEvent)](https://developer.android.com/reference/android/view/View.html#onTouchEvent(android.view.MotionEvent)) - Called when a touch screen motion event occurs.
* [onFocusChanged(boolean, int, Rect)](https://developer.android.com/reference/android/view/View.html#onFocusChanged(boolean,%20int,%20android.graphics.Rect)) - Called when the view gains or loses focus.