**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.

Type Casting

Assigning a value of one type to a variable of another type is known as Type Casting.

Type casting is classified into two types,

**Widening Casting** (automatically) - converting a smaller type to a larger type size  
byte -> short -> char -> int -> long -> float -> double

**Narrowing casting** - must be done manually by placing the type in parentheses in front of the value

Double ->float ->long ->int ->char ->short ->byte

**JSON**

JSON stands for JavaScript Object Notation. It is an independent data exchange format and is the best alternative for XML.

This class can coerce values to another type when requested.

* When the requested type is a boolean, strings will be coerced using a case-insensitive comparison to "true" and "false".
* When the requested type is a double, other [Number](https://developer.android.com/reference/java/lang/Number.html) types will be coerced using [Number#doubleValue()](https://developer.android.com/reference/java/lang/Number.html#doubleValue()). Strings that can be coerced using [Double#valueOf(String)](https://developer.android.com/reference/java/lang/Double.html#valueOf(java.lang.String)) will be.
* When the requested type is an int, other [Number](https://developer.android.com/reference/java/lang/Number.html) types will be coerced using [Number#intValue()](https://developer.android.com/reference/java/lang/Number.html#intValue()). Strings that can be coerced using [Double#valueOf(String)](https://developer.android.com/reference/java/lang/Double.html#valueOf(java.lang.String)) will be, and then cast to int.
* When the requested type is a String, other non-null values will be coerced using [String#valueOf(Object)](https://developer.android.com/reference/java/lang/String.html#valueOf(java.lang.Object)). Although null cannot be coerced, the sentinel value [JSONObject#NULL](https://developer.android.com/reference/org/json/JSONObject.html#NULL) is coerced to the string "null".

This class can look up both mandatory and optional values:

* Use getType() to retrieve a mandatory value. This fails with a JSONException if the requested name has no value or if the value cannot be coerced to the requested type.
* Use optType() to retrieve an optional value. This returns a system- or user-supplied default if the requested name has no value or if the value cannot be coerced to the requested type.

**Adapter**

An Adapter object acts as a bridge between an [AdapterView](https://developer.android.com/reference/android/widget/AdapterView.html) and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a [View](https://developer.android.com/reference/android/view/View.html) for each item in the data set.

**ArrayAdapter**

The adapter provide views for an [AdapterView](https://developer.android.com/reference/android/widget/AdapterView.html), Returns a view for each object in a collection of data objects you provide, and can be used with list-based user interface widgets such as [ListView](https://developer.android.com/reference/android/widget/ListView.html) or [Spinner](https://developer.android.com/reference/android/widget/Spinner.html).

The array adapter creates a view by calling [Object#toString()](https://developer.android.com/reference/java/lang/Object.html#toString()) on each data object in the collection you provide, and places the result in a TextView.

**CursorAdapter**

* Adapter that exposes data from a [Cursor](https://developer.android.com/reference/android/database/Cursor.html) to a [ListView](https://developer.android.com/reference/android/widget/ListView.html) widget.
* The Cursor must include a column named "\_id" or this class will not work. Additionally, using [MergeCursor](https://developer.android.com/reference/android/database/MergeCursor.html) with this class will not work if the merged Cursors have overlapping values in their "\_id" columns.