**Encapsulation----Binding properties & behaviors**

Encapsulation ensures data integrity and reusability. Allows programmer to hide instance data and methods from users. It's a protective barrier that protects the code and data from being randomly accessed by other code. Implemented using getter and setter methods.

**Abstraction---Is a process of highlighting the set of services and hiding the implementation in child class**

Abstraction is basically the process of hiding certain details and only showing certain essential details of an object. Abstraction is removing important concepts of something. It's nothing but, instead of concentrating on the main implementation of something we just concentrate on the abstract apart of it. Implemented using abstract classes and abstract methods.

**Inheritance**

Inheritance is a process in which one object acquires all the properties of the parent object. The keyword extends is used to inherit the properties of the parent class. The class which is inherited is called a parent class or a super class and the class that derives all the properties of the parent class is a child class or a sub class.

**Polymorphism**

It is the ability of an object to take on many forms. It is implemented through method overloading and method overriding.

**Method Overloading**: Same method name, but different parameters within the same class

**Method Overriding**: Same method name, same number of parameters, but in different classes

**Types of Polymorphism**

There are two types of polymorphisms:

Run-Time Polymorphism(method overriding)

Compile time Polymorphism(method overloading)

**Value Type**

int a = 5;

This is a value type. When we declare a variable and assign a value to it, the variable itself holds the value.

**Reference Type**

Box b = new Box();

In the above code an instance of the Box class is declared and the Box class has some properties. The instance 'b' does not hold any value, instead it holds the reference to the physical object created in the heap memory. Hence it is a reference type.

**Call by Value**

In java everything is call by value

**Abstract class: In terms of preference we**

When a class contains abstract methods, then it must be declared as abstract. An abstract method contains only method declaration but does not contain method definition.

An abstract class must contain at least one abstract method. It can also contain concrete methods.

We cannot create object of an abstract class. If we create an object for the abstract class and try to invoke the abstract method, the JVM will not understand which code to run as, abstract methods do not have implementation. Hence, the compiler restricts the programmer from creating an object for the abstract class.

How to access the concrete methods of an abstract class?

The concrete methods of an abstract class can be accessed in two way:

1. adding a static key word to the method. By doing this we can access the method with the class name.

2. extending the abstract class. by doing this we can create an object of the class that extends the abstract class and access the concrete methods of the abstract class.

**Interfaces** Interface is a reference type and is almost similar to abstract classes. An interface contains only abstract methods. Interface contains behaviors that the class implements. If a class wants to define the methods of the interface, it must implement the interface using 'implements' keyword. For example, class A implements A

All the fields in a interface are public, static and final and all the methods are public

**Collections**

Collection is an object that groups multiple elements into a single unit. They are used to store, retrieve, manipulate and communicate data.

The core collection interfaces are Set, List, Map, Queue and Deque

List Interface:

* accepts duplicate elements
* order based on the insertion order
* accepts null values

Set Interface:

* Does not accept duplicate elements
* HashSet : Returns elements in a random order
* TreeSet: Returns elements in a sorted format
* LinkedHashSet: Returns elements in the order they were inserted

Map Interface:

* Contains pairs of Keys and values
* acceptsduplicate values but the keys must be unique
* If we try to insert elements with duplicate keys, the recently added value becomes the value of the key i.e., the previous value of the key is overridden

For example,

if we insert as follows

map.put(1,"manasa");

map.put(1,"joshi");

The elements stored in the Map is as follows:

1, joshi

* HashMap: Prints elements in a random order
* TreeMap: Prints elements in a sorted order of keys
* LinkedHashMap: Prints elements in the order they were inserted

**Difference between HashSet, TreeSet and Linked HashSet**

**HashSet** internally implements Hashtable . The elements in the HashSet are not ordered

**TreeSet** internally implements red-black tree algorithm. The elements in TreeSet are sorted in ascending order.

**LinketHashSet** internally implements hash table with a linked list running through it. The elements are stored in the insertion order.

**How does a HashMap work?**

HashMap internally implements hashtable. When the put( ) method is called, a hash value for the key is generated. Using this hash value an index number is generated so that the value can be added to the table. The index is calculated as follows:

index = hashvalue % (n-1); where n is the length of the table that is 16

Once the index is created, a linkedlist that contains the key, hash value of the key, value assigned to the key, a pointer to the next node is created and assigned to that index.

If the index created is already occupied by a linkedlist then, the pointer of the 1st list points to the next list.

When the get( ) method is called, a hash value for the key is generated. Using this hash value an index number is generated same as it works for the put( ) method. Then the control goes to that particular index and check whether the hash code generated and the hash code present in the list are equal. If they are equal, it checks if the key provided and the key present in the list are equal. If they are equal, it then returns the value that is present in the list.

What if there are multiple lists at the index that was generated? In this case, the hash code is compared with 1st list, if it does not match then, the control is shifted to the next list and, the hash code is checked, if there is a match then the key values are checked and if there is a match found, the value presen

getWIndowHandles() will give all window list of Strings

what is the project about ?

Sorry getWindowHandles returns a Set of windows, set does not contain duplicate

Driver.window.swtichTo()

**Multi-threading**

**Multithreading in java** is a process of executing multiple threads simultaneously

**Thread:**  It is a smallest unit of process. Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory location. Every JAVA program has atleast one thread that is the main thread.

**Uses of threads:**

1. To make the complete use of CPU

2. Creating Web applications using Servlets

3. To develop a gaming application

**How can we implement threads?**

1. By extending the Thread class

2. By implementing the Runnable interface

**Life Cycle of a Thread**

**New**: A thread begins its life cycle in the new state. It remains in this stage until the program starts the thread.

**Runnable**: Once a thread it started it enters into the runnable state. The thread in this state is said to be ready for execution.

**Running**: When a run method is called the Thread in the runnable state enters the running state. Here the Thread is executed.

**Waiting**: During the execution if the program requires an I/O operation to be performed or the resource that is to be accessed by the Thread is held by another thread then, the Thread enters the waiting state.

**Dead**: Once the Thread has completed its execution it goes to the dead state.