Difference between Abstract Class and Interface

Abstract class	Interface
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
An abstract class can extend another class and can implement multiple interfaces.	An interface can extend another interface only.
7) An abstract class can be extended using keyword "***extends***".	An interface class can be implemented using keyword "***implements***".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)Example:
public abstract class Shape {
public abstract void draw();
}
	Example:
	public interface Drawable {
	void draw();
	}

Difference between Array List and Linked List

ArrayList	LinkedList
ArrayList internally uses a dynamic array (growable/resizable) to store the elements.	LinkedList internally uses a doubly linked list to store the elements.
Data is stored in the form of array.	Data is stored in the form of node.
Initial Capacity and Incremental Capacity is applicable.	Initial Capacity and Incremental Capacity is not applicable.
Involves shift operations. That is manipulation with ArrayList is slow because it internally uses an array and If any element is removed from the array, all the bits are shifted in memory.	Does not involves any shift operations. That is manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
Has three overloaded constructors.	Has two overloaded constructors.
Memory is continuous.	Memory may not be continuous.
An ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data. (Insertion or removal of data in between)
Consumes less memory.	Consumes more memory.

Difference between Array List and Vector

ArrayList	Vector
ArrayList is not synchronized. That is ArrayList is multi-threaded.	Vector is synchronized. That is Vector is single threaded.
If the total number of elements exceeds than its capacity, the incremental capacity of the ArrayList is **((Current Capacity * 3)/2) + 1**	Vector increments 100% which means it doubles the array size if the total number of elements exceeds than its capacity. That is **Current Capacity * 2**
ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a legacy class. It is present since JDK 1.0.
ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
ArrayList uses the Iterator interface to traverse the elements.	A Vector can use the Iterator interface or Enumeration interface to traverse the elements.
Has three overloaded constructors.	Has four overloaded constructors.
Cannot control the growth. That is the incremental capacity is fixed.	Can control the growth. That is the incremental capacity can be explicitly mentioned by the programmer. It is done using one of the overloaded constructors. **public vector (int initialCapacity, int incrementalCapacity)**

Difference between Comparable and Comparator

Comparable	Comparator
Comparable affects the original class, i.e., the actual class is modified.	Comparator doesn't affect the original class, i.e., the actual class is not modified.
Comparable provides compareTo() method to sort elements.	Comparator provides compare() method to sort elements.
Comparable is present in java.lang package.	A Comparator is present in the java.util package.
We can sort the list elements of Comparable type by Collections.sort(List) method.	We can sort the list elements of Comparator type by Collections.sort(List, Comparator) method.

Difference between final, finally and finalize

final	finally	finalize
final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	finally is used to place important code, it will be executed whether exception is handled or not.	finalize is used to perform clean up processing just before object is garbage collected.
final is a keyword.	finally is a block.	finalize is a method.

Difference between HashMap and Hashtable

HashMap	Hashtable
HashMap is non-synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized. It is thread-safe and can be shared with many threads.
HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value.
HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
HashMap is fast.	Hashtable is slow.
We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
HashMap is traversed by Iterator.	Hashtable is traversed by Enumerator and Iterator.
Iterator in HashMap is fail-fast.	Enumerator in Hashtable is not fail-fast.
HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Difference between StringBuffer and StringBuilder

StringBuffer	StringBuilder
StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

Difference between Object and Class

Object	Class
Object is an instance of a class.	Class is a blueprint or template from which objects are created.
Object is any real-world entity that is physically present or that can be experienced.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{}
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.

Difference between String and StringBuffer

String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you concatenate strings.
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Difference between throw and throws

Throw	throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException, SQLException.

Difference between == and equals()

==	equals()
It is an operator.	It is a method.
In case of non-primitive, == operator compares the reference (address). That is == checks if both objects point to the same memory location.	In case of non-primitive, the .equals() method is used for content comparison. That is, .equals() evaluates to the comparison of values in the objects.
No such things with related to ==	If a class does not override the equals method, then by default it uses equals(Object o) method of the closest parent class that has overridden this method.
== is used mostly to compare the primitive values.	Equals() method is used to compare the contents of the object.

Difference between static and dynamic binding

Early/Static Binding	Late/Dynamic Binding
Occurs at compile time.	Occurs at run time.
Used to resolve overloaded methods.	Used to resolve overridden methods.
Decision about which implementation has to be executed is taken by the compiler based on the arguments.	Decision about which implementation has to be executed is taken by the JVM based on the invoking objects.
private, static and final methods are resolved by static binding as they cannot be overridden.	Virtual/instance/non-static methods are resolved by dynamic binding as they can be overridden.
Actual object us not used to locate the method. Instead, they type information of the variable is used to locate the method.	Actual object is used to locate the method.
Possible even if there is no [is-a] relationship.	Possible only if there is a [is-a] relationship.
Method's signature has to vary in the event of using method overloading also the method implementation may vary.	Method's signature must not vary where in the method implementation must definitely vary in case of method overriding.