Unit 9:

Understanding Core Packages

Core Java Packages:

1. Java Lang Package
2. Java Util Package

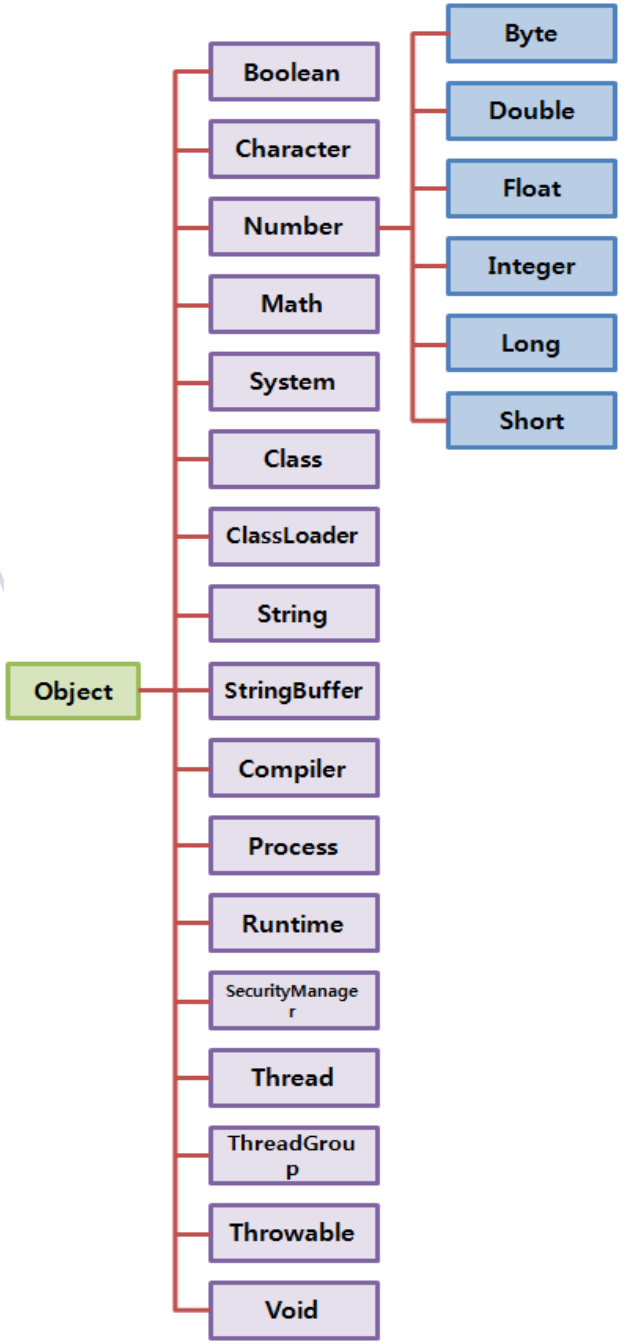
1. java.lang package :

- ❖ Contains classes that are fundamental to the design of the Java programming language.

- ❖ It is not required to import java.lang package in our program because it is available by default to every java program.

- ❖ The following are some of important classes present in java.lang package:

1. Object class
2. Math class
3. String class
4. StringBuffer class
5. StringBuilder class
6. Wrapper Classes
7. Autoboxing and AutoUnboxing



Java.lang.Math Class :

- The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

- **Declaration :**

```
public final class Math extends Object
```

Basic Math methods:

Method	Description	Arguments
abs	Returns the absolute value of the argument	Double, float, int, long
round	Returns the closed int or long (as per the argument)	double or float
ceil	Math ceil function in Java returns the smallest integer that is greater than or equal to the argument	Double
floor	Java floor method returns the largest integer that is less than or equal to the argument	Double
min	Returns the smallest of the two arguments	Double, float, int, long
max	Returns the largest of the two arguments	Double, float, int, long
random	returns a random number between 0.0 (inclusive), and 1.0 (exclusive)	
pow	It returns the value of first argument raised to the power to second argument.	Double, int

Logarithmic Math methods:

Method	Description	Arguments
log10	It is used to return the base 10 logarithm of a double value.	Double
exp	It returns the value ex, where e is the base of the natural logarithms.	Double

Trigonometric Math methods:

Method	Description	Arguments
sin	It is used to return the trigonometric Sine value of a Given double value.	Double, Eg. an angle, in radians
cos	It is used to return the trigonometric Cosine value of a Given double value.	Double
tan	It is used to return the trigonometric Tangent value of a Given double value.	Double

Wrapper class and associated methods:

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Autoboxing:

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Unboxing:

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Need of Wrapper Classes :

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).

- The classes in java.util package handles only objects and hence wrapper classes help in this case also.

- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

- An object is needed to support synchronization in multithreading.

- The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

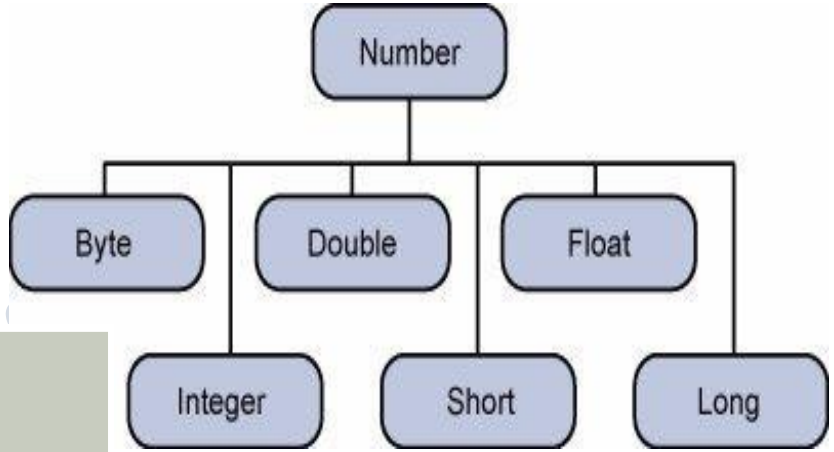
Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Java Numbers Class :

- Java Number class is an abstract class which is placed in java.lang package.
- All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.

- Java Number class provides methods to convert the represented numeric value to byte, double, float, int, long, and short type.

- The various Java Number methods are as follows :



SN	Modifier & Type	Method	Description
1)	Byte	byteValue()	It converts the given number into a byte type and returns the value of the specified number as a byte.
2)	abstract double	doubleValue()	It returns the value of the specified number as a double equivalent.
3)	abstract float	floatValue()	It returns the float equivalent value of the specified Number object.
4)	abstract int	intValue()	It returns the value of the specified number as an int.
5)	abstract long	longValue()	It returns the value of the specified number object as long equivalent.
6)	short	shortValue()	It returns the value of the specified number as a short type after a primitive conversion.

Java Double Class :

- The Double class generally wraps the primitive type double into an object.
- An object of Double class contains a field with the type Double.
- Following are methods of Double class :

Method Name	Description	Example
1. toString() :	Returns the string corresponding to the double value.	double b = 55.05; Double x = new Double(b); String s=Double.toString(x);
2. valueOf() :	returns the Double object initialised with the value provided.	String s="45.55"; Double d=Double.valueOf(s);
3. parseDouble() :	returns double value by parsing the string. Differs from valueOf() as it returns a primitive double value and valueOf() return Double object.	String s="45.55"; Double d=Double.parseDouble(s);

4. equals() :	Used to compare the equality of two Double objects. This methods returns true if both the objects contains same double value.	If(double1.equals(double2))
		//true
		else // false

5. **compareTo()**	Used to compare two Double objects for numerical equality. Returns a value less than 0,0,value greater than 0 for less than,equal to and greater than.	If(double1.compareTo(double2))
		//true
		else // false

6. compare() :	Used to compare two primitive double values for numerical equality.	If(compare(double1,double2)==0) //equal
7. byteValue() :	returns a byte value corresponding to this Double Object.	byte b=double1.byteValue();
8. shortValue() :	returns a short value corresponding to this Double Object.	short s=double1.shortValue();
9. intValue() :	returns a int value corresponding to this	int i=double1.shortValue();

Java Float Class :

- The Float class wraps a value of primitive type float in an object.
- An object of type Float contains a single field whose type is Float.
- Following are methods of Float class :

Method Name	Description	Example
1. toString() :	Returns the string corresponding to the float value.	float b = 55.05; Float x = new Float (b); String s= Float.toString(x);
2. valueOf() :	returns the Float object initialised with the value provided.	String s="45.55"; Float d= Float.valueOf(s);
3. parseFloat() :	returns float value by parsing the string. Differs from valueOf() as it returns a primitive float value and valueOf() return Float object.	String s="45.55"; Float d= Float.parseDouble(s);

4. equals() :	Used to compare the equality of two Float objects. This methods returns true if both the objects contains same float value.	If(float1.equals(float2))
		//true
		else // false

5. **compareTo()**	Used to compare two Float objects for numerical equality. Returns a value less than 0,0,value greater than 0 for less than,equal to and greater than.	If(float1.compareTo(float2))
		//true
		else // false

6. compare() :	Used to compare two primitive float values for	If(compare(float1,float2)==0) //equal

Java Boolean Class :

- The Boolean class wraps a value of primitive type boolean in an object.
- An object of type Boolean contains a single field whose type is Boolean.
- Following are methods of Boolean class :

1. parseBoolean(String s) :

- Used to parse the string argument as a boolean. It returns a boolean value either true or false. If the string argument is not null and is equal, ignoring case, to the string "true", it returns true.
- It takes a string argument that contains the boolean representation to be parsed.

2. booleanValue() :

- It is used to get boolean value of this Boolean object as a boolean primitive. It returns primitive type boolean value of the Boolean object.

3. valueOf(boolean b)

- It is used to get Boolean instance representing the specified boolean value. If the specified boolean value is true, this method returns Boolean. **TRUE**; if it is false, this method returns Boolean. **FALSE.**

4. valueOf(String s) :

- It is used to get a Boolean with a value represented by the specified string. It takes a string argument and returns a Boolean value represented by the specified argument.

5.equals(Object obj) :

- It is used to check if two boolean objects represent the same value. This method is used to compare two boolean values and returns true if both are equal, false otherwise.

Java Character Class :

- The Character class is a wrapper that is used to wrap a value of the primitive type char in an object. An object of class Character contains a single field whose type is char.

- Following are methods of Character class :

1. isLetter(char ch) :

- It is used to determine if the specified character is a letter or not. It returns a boolean value either true or false. If the character is a letter then it returns true else false. It takes a single char type argument

2. isDigit(char ch) :

- It is used to determine if the specified character is a digit or not. It returns true if the specified character is a digit, false otherwise. It takes a single char type argument.

3. isWhitespace(char ch) :

- It is used to determine whether the specified character is a white space or not. It returns true if the character is a Java whitespace, false otherwise.

4. isUpperCase(char ch):

- It is used to determine if the specified character is an uppercase character or not. It returns true if the specified character is upper case letter, false otherwise. It takes a single char type argument.

5. isLowerCase(char ch) :

- It is used to check whether the specified character is lowercase letter or not. It returns true if the specified character is in lowercase, false otherwise.

6. toUpperCase(char ch):

- It is used to convert the character argument to uppercase. It returns a character after converting to the uppercase. It takes a single argument of char type.

7. toLowerCase(char ch) :

- It is used to convert the character argument to lowercase letter. It returns a character after converting to the lowercase. It takes an single character type argument.

Java Byte Class :

- The Byte class is a wrapper class which is used to wraps a value of primitive type byte in an object. An object of type Byte contains a single field whose type is byte.

Following are the methods of Byte class :

1. toString()
2. valueOf() : It returns a Byte instance representing the specified byte value.
3. byteValue() : It is used to get a primitive type byte value from Byte object.
4. shortValue() : This method returns the value of this Byte as a short after a widening primitive conversion.
5. doubleValue() : It returns the value of this Byte type as a double type after a widening primitive conversion.
6. floatValue() : This method is used to get value of this Byte type as a float type after a widening primitive conversion.
7. equals() : It is used to compares an object to the specified object. It returns true if objects are same; false otherwise.
8. compare() : It is used to compare two byte values numerically. The value returned is identical to what would be returned by.

Data Structures :

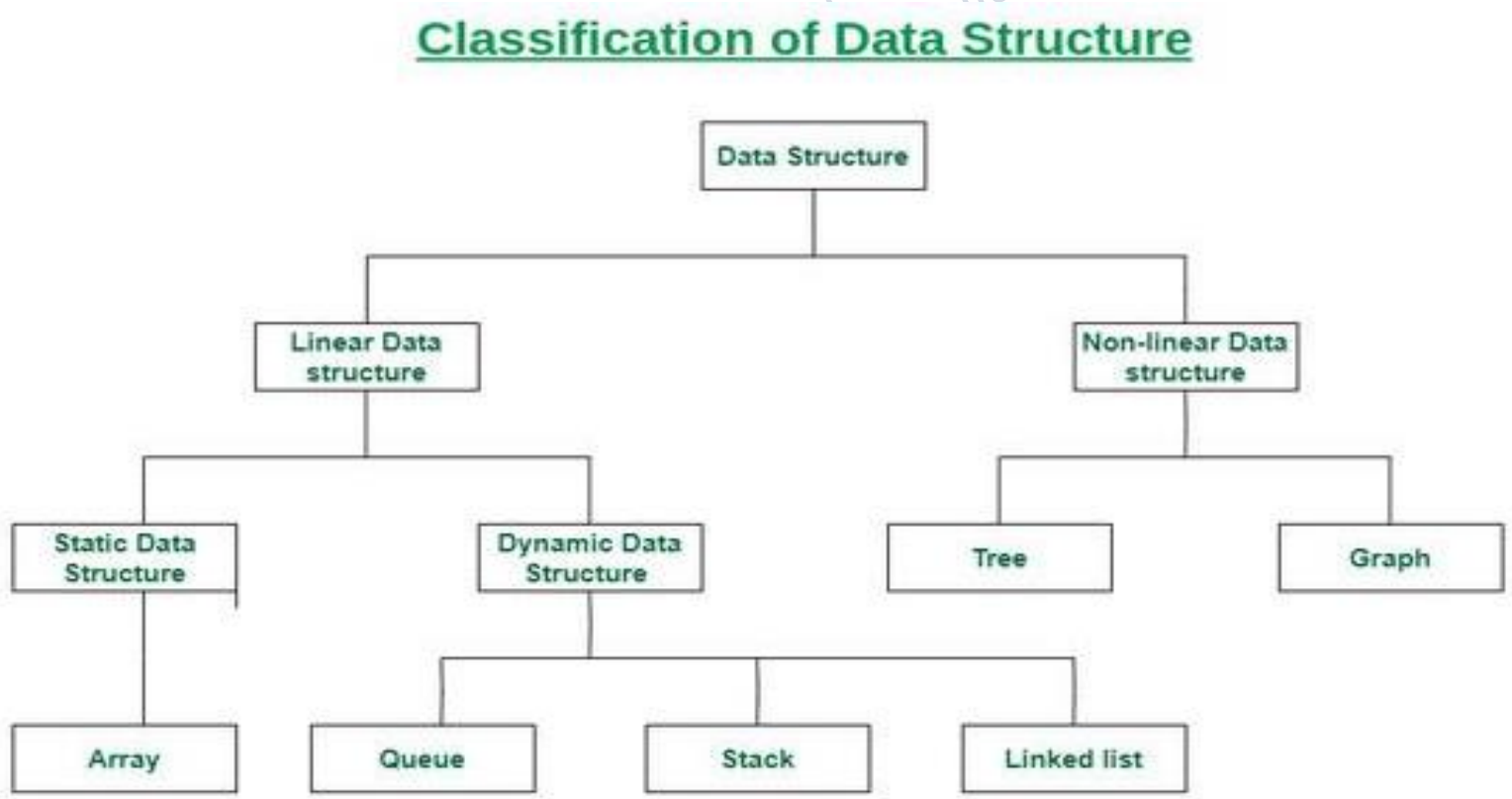
- A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Linear data structures

In linear data structures, the elements are arranged in sequence one after the other.

Non-linear data structures

The elements in non-linear data structures are not in any sequence. Instead, they are arranged in a hierarchical manner where one element will be connected to one or more elements.



2. java.util package :

- Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Following are the Important Classes in Java.util package :

1. Vector: The Vector class implements a growable array of objects.
2. Stack: The Stack class represents a last-in-first-out (LIFO) stack of objects.
3. Dictionary<K,V>: The Dictionary class is the abstract parent of any class, such as Hashtable, which maps keys to values.
4. Hashtable<K,V>: This class implements a hash table, which maps keys to values.
5. Random: An instance of this class is used to generate a stream of pseudorandom numbers.
6. Enumerations : ***(enum for short)*** in Java is a special data type which contains a set of predefined constants.

1. Vector:

- The Vector class implements a growable array of objects.
- Vectors fall in legacy classes, but now it is fully compatible with collections.
- It is found in java.util package and implement the List interface, so we can use all the methods of the List interface.
- Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
- It also maintains an insertion order like an ArrayList. Still, it is rarely used in a non-thread environment as it is synchronized, and due to this, it gives a poor performance in adding, searching, deleting, and updating its elements.

Vector Constructors :

- Vector class supports four types of constructors. These are given below:

1. Vector() :

- It constructs an empty vector with the default size as 10.

Syntax:

```
Vector<E> vector  = new Vector<E>();
```

2. Vector(int initialCapacity) :

- It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.

Syntax:

```
Vector<E> vector  = new Vector<E>(int  size);
```

3. Vector(int initialCapacity, int capacityIncrement) :

- It constructs an empty vector with the specified initial capacity and capacity increment.

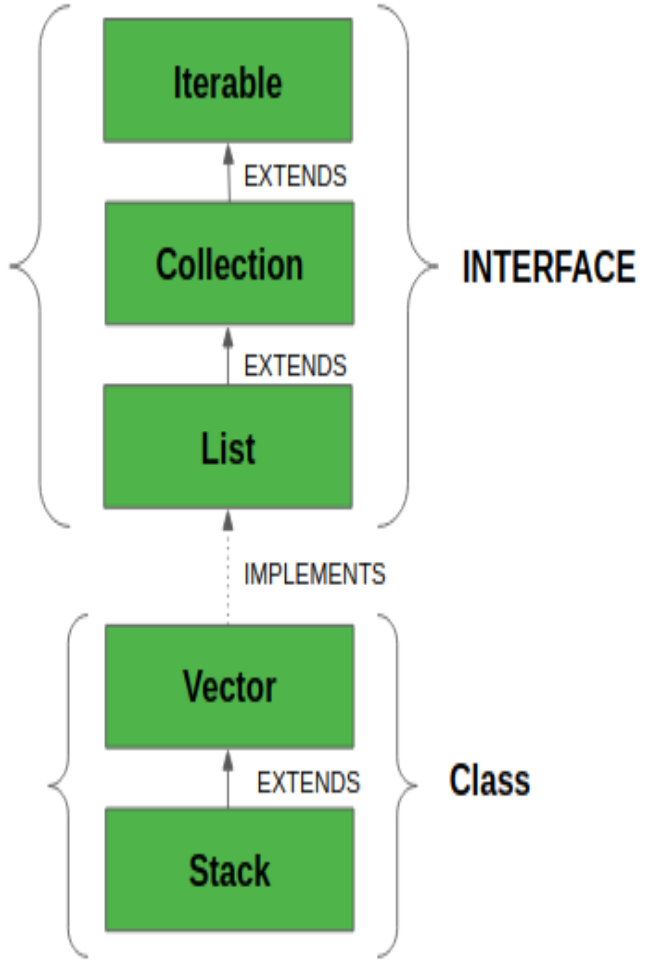
Syntax:

```
Vector<E> vector  = new Vector<E>(int  size, int  incr);
```

4. Vector(Collection<? extends E> c)

- It constructs a vector that contains the elements of a collection c.

Note : Here, E is the type of element.



2. Stack:

- Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

- Java Collection framework provides a Stack class that models and implements a Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.

How to Create a Stack?

- To create a stack, we must import java.util.stack package and use the Stack() constructor of this class.
- The below example creates an empty Stack.

```
Stack<E> dataStack = new Stack<E>();
```

- Here E is the type of Object.

Stack Methods :

1. push() : To add an element to the top of the stack, we use the push() method.

2. pop() : To remove an element from the top of the stack, we use the pop() method.

3. peek() : The peek() method returns an object from the top of the stack.

4. search() : To search an element in the stack, we use the search() method. It returns the position of the element from the top of the stack.

3. Dictionary :

- In Java, Dictionary is the list of key-value pairs. We can store, retrieve, remove, get, and put values in the dictionary by using the Java Dictionary class.

- Since this class is abstract, we won't work with it directly. Dictionary has a direct child class Hashtable. So for creating a dictionary in Java you can use Hashtable.

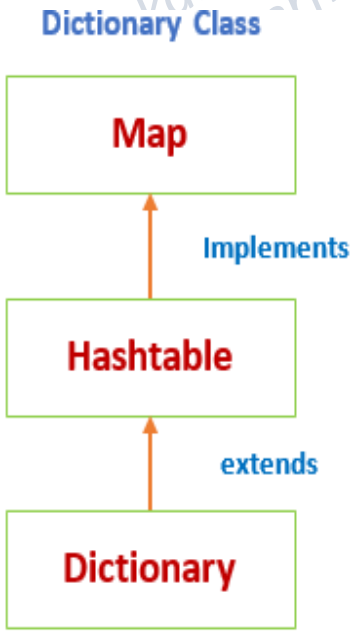
Following are the important points about Dictionary :

1. In this class every key and every value is an object.
2. In this class object every key is associated with at most one value.

Class constructors

Dictionary()

- The Dictionary class hierarchy is as follows:



Creating Dictionary:

Syntax:

```
Dictionary<Key, Value> phoneBook = new Hashtable<>();
```

Eg.

```
Dictionary<String, String> phoneBook = new Hashtable<>();
```

Methods :

1. put(K key, V value) : java.util.Dictionary.put(K key, V value) adds key-value pair to the dictionary.

2. get(Object key) : java.util.Dictionary.get(Object key) returns the value that is mapped with the argumented key in the dictionary.

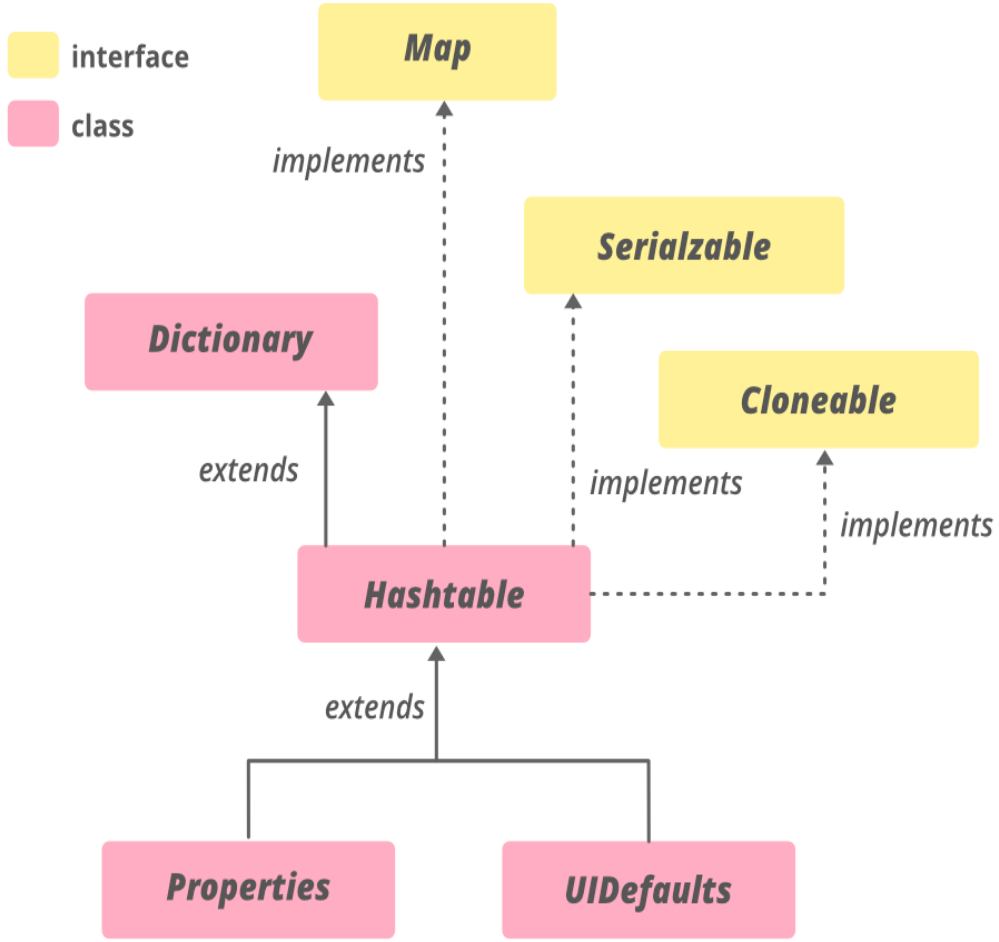
3. size() : java.util.Dictionary.size() returns the no. of key-value pairs in the Dictionary.

4. remove(Object key) : java.util.Dictionary.remove(Object key) removes the key-value pair mapped with the argumented key.

4. Hashtable:

- Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.
- It is similar to HashMap but is synchronized.
- At a time only one thread is allowed to operate the Hashtable's object. Hence it is thread-safe.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.

The Hierarchy of Hashtable is as follows:



Declaration:

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable

Type Parameters:

•K – the type of keys maintained by this map
•V – the type of mapped values

Constructors:

1. Hashtable():

- It creates an empty hashtable having the initial default capacity (11) and load factor(0.75).

2. Hashtable(int initialCapacity):

- This constructs a new, empty hashtable with the specified initial capacity and default load factor (0.75).

3. Hashtable(int initialCapacity, float loadFactor):

- This constructs a new, empty hashtable with the specified initial capacity and the specified load factor.

4. Hashtable(Map<? extends K,? extends V> t):

- This constructs a new hashtable with the same mappings as the given Map.

Methods :

1. put(K key, V value) :This method maps the specified key to the specified value in this hashtable.
2. get(Object key) :This method returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
3. size() : This method returns the number of keys in this hashtable.
4. remove(Object key) :This method removes the key (and its corresponding value) from this hashtable.

5. Enumerations :

- Enumerations was added to Java language in JDK5.
- Enumeration means a list of named constant.
- It is created using enum keyword.
- Each enumeration constant is public, static and final by default.
- Enumerations can have Constructors, instance Variables, methods and can even implement Interfaces.
- Enumerations are not instantiated using new keyword.
- Enum can be easily used in switch.

Define and Use an Enumeration:

1. An enumeration can be defined simply by creating a list of enum variable.

```
enum Subject{
     BASIC_JAVA,
     OOP_JAVA,
     ADVANCED_JAVA
}
```

2. Identifiers ***BASIC_JAVA***, ***OOP_JAVA*** and ***ADVANCED_JAVA*** are called **enumeration constants**. These are public, static and final by default.

3. Variables of Enumeration can be defined directly without any **new** keyword.

6. Random:

- Random class is part of java.util package.
- An instance of java Random class is used to generate random numbers.
- This class provides several methods to generate random numbers of type integer, double, long, float etc.
- Random number generation algorithm works on the seed value. If not provided, seed value is created from system nano time.
- If two Random instances have same seed value, then they will generate same sequence of random numbers.
- Java Random class is thread-safe, however in multithreaded environment it's advised to use **java.util.concurrent.ThreadLocalRandom** class.
- Random class instances are not suitable for security sensitive applications, better to use **java.security.SecureRandom** in those cases.

- **Constructors:**

1. Random(): ***Creates a new random number generator.***
2. Random(long seed): ***Creates a new random number generator using a single long seed.***