**Core Java**

* **public static void main()**

1. Static -> do not have to create an Object of the class
2. Void -> will not return anything.

* **Constructor** : cannot be declared as final since it cannot be inherited.
* **Static Variable** - > These are class level variable and it is initialized only once. And occupy only one storage space.

static String company\_name = "StudyTonight"; All employee have its unique name and employee id but company name will be same all 100 employee.

* **Static Method** : It can be accessed without creating an object.

public static void square(int x) { System.out.println(x\*x);

}

public static void main (String[] arg)

{ square(8)

}

* **Static Block** : Used to initialize static data member, it is executed before main method.
* **Static Class** : A static class cannot access non-static members of the Outer class.
* **Final Variable :** Final variable once assigned a value can never be changed.If we try to change we will get compilation error(CompileTime error)
* **Final Method :** If we make method as final it cannot be overridden**. It can be inherited like**
* **Final Class ->** If we make as final class it cannot be extended.
* **Abstract Class** -> Can have abstract method and concrete method. Abstract class classname{ // it can only be declared cannot initialized}. Normal method also can be declared inside abstract class.
* String == ( refer to the object memory location) and equals refer to the content in the object. String obj =”test” and String obj1=”test1” obj==obj1 it return false both are different object location but can be true by String obj1=obj;
* **String** – Immutable once declared we cannot change the size or capacity
* **StringBuffer** : Muttable , It is thread safe,
* **String Builder** - > It is not synchorinzed so it is not thread safe.
* Exception – Abnormal flow of executaion of a program, checked exception are compile time exception and unchecked is runtime exception
* Thread : **Implements Runnable** , here we created a thread and need to have run method and while creating a object we need to create new Thread .

**class** MyThread **implements** Runnable

{

**public** **void** run()

{

System.***out***.println("Pradeep Here");

}

}

**public** **class** Thread1 {

**public** **static** **void** main(String arg[])

{

MyThread t1 = **new** MyThread();

**Thread t = new Thread(t1); // create new thread to access the run method.**

t.start();

}

}

* **Extends Thread :** override the **run()** and then use the **start()** method to start and run the thread.

**class** MyThreads **extends** Thread

{

**public** **void** run()

{

System.***out***.println("Pradeep Here");

}

**public** **void** display()

{

System.***out***.println("Project Lead");

}

}

**public** **class** ThreadExtends {

**public** **static** **void** main(String arg[])

{

MyThreads t = **new** MyThreads();

t.start();

t.display();

}

}

* **Difference between extends thread and implements Runnable**

1. Extends Thread class, each of your thread creates unique object and associate with it.
2. Implements Runnable, it shares the same object to multiple threads.

**Generics**

1. A class or interface that operates on parameterized type is called **Generic**.
2. Automatically works with all types of data(Integer, String, Float etc).

**class** GenericClass<Student>

{

Student name; //an object of type T is declared

GenericClass(Student names) //constructor

{

name = names;

}

**public** Student getName()

{

**return** name;

}

**public** **static** **void** main (String[] args)

{

GenericClass < String> sob = **new** GenericClass("Pradeep"); //instance of String type Gen Class.

String str = sob.getName();

System.***out***.println(str);

}

}

**Collections :**

1. **Arraylist** use ascending order sorting type, if we want to make it as decending order we can use collections.reverseOrder()
2. **Vector** similar to arraylist, but it is synchronized and used enumeration for retrieving the object and use elements().addElement(),elementAt(),firstElement(),lastElement(),removeAllElement()

B ) **Linked List** - It can contain duplicate elements and is not synchronized..

C) HastSet – Implements set interface doesnot maintain any order of elements.

D) LinkedHashSet – Maintains order while inserting what place we mentioned it will maintain that order only.

E) TreeSet - Stores element in ascending order, Tree structure to store elements,

**Map :** stores data in key and value association. Both key and values are objects. key must be unique but the values can be duplicate

1. Hash Map - \* It may have one null key and multiple null values

* It maintains no order

hm.put(100,"Amit");

 for(Map.Entry m:hm.entrySet())

System.out.println(m.getKey()+" "+m.getValue());

1. Tree Map : provides key/value pairs in sorted order
2. HashTable : stores key/value pair , but doesnot allow either key or value as null.

**Hashtable< String,Integer> ht = new Hashtable< String,Integer>()**;

ht.put("a",new Integer(100));

Set st = ht.entrySet();

Iterator itr=st.iterator();

while(itr.hasNext())

{

Map.Entry m=(Map.Entry)itr.next();

System.out.println(itr.getKey()+" "+itr.getValue());

}

1. Serialization : Sequence of bytes that includes the object's data and also data stored in the object. classes **ObjectInputStream** and **ObjectOutputStream**  are used for serialization and desearilzation . mostly used during file writing and reading.
2. Object Cloning in Java - **java.lang.Cloneable interface** must be implemented by the class. Clone() copies the value of an object to another
3. Immutable class – class can be created as immutable using final.
4. ConcurrentHashMap - Does not allow NULL values . It is used only in single threaded concepts. It is a thread safe so code can access one thread at a time.
5. Singleton Class - accessed directly without need to instantiate the object of the class. class have its constructor as private and have a static instance
6. What is immutable object? Can you write immutable object?

Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object result in new object. For example is String is immutable in Java. Mostly Immutable are also final in Java. Accessing elements are faster with ArrayList, because it is index based. But accessing is difficult with LinkedList. It is slow access. –

But insertion and deletion is much faster with LinkedList, because if you know the node, just change the pointers before or after nodes.