Chapter 9 Important Notes : LIFE AND DEATH OF AN OBJECT

- First thing we will talk about it is the stack and the heap where things live the 2 areas of memory okay now when the JVM starts it gets a chunk up of memory from the os and use it to run the written java code okay now in the stack we will have method invocations and local variables live there and in the heap or we can say in the garbage collectable heap where all objects live there
- There is difference between instance variable and local variables the instance variables is the variables that are declared inside a class but not inside a method they are the attributes or the data members of the class the instance variables lives inside the object they belongs to in the other hand we have the local variables and this ones are declared but inside the method including the method parameters and they are temporary and live only as long as the method on the stack
- Methods are stacked when you call a method it is pushed into the stack and the method on the top of the stack is always the currently executing method
- What about if the local variables (that pushed in the stack) are objects? Remember that any non primitive variable holds a reference to the object doesnot have the object it self the object it self is created in the heap and the only reference variable which is the local one here will still be existed in the stack too cause it is not object it just reference variable to it
- If local variables live on the stack so where the instance variables live?
  They are live in the heap inside the object itself and that cause when making space in the heap to be good for the object the space here its size depends on the instance variables cause they stored in this place with their objects to can store this object special values to this ones
  - What about if the class instances are non primitive ones that means we will go to store the object normally with the size of total instance variables I have and give attention here that the instance variable which is non primitive it will be stored in memory but its only reference type cause there Is no object created until now
- Remember the three steps of object declaration and assignment: declare a reference variable, create an object, and assign the object to the reference.
- We now will start speaking about the class constructor and the constructor look and feel a lot like the normal class methods but it is a special method it start working when you say new in the creation of the objects that means the constructor is the code that runs when you create an object you can write a constructor for your class but if you do not do that the compiler writes one for

you and it is the default constructor , constructor when we write it , it having the same class name exactly and you do not define its return type like other methods , the benefit of the constructor here is to go and help you in creating really existed object in the heap and give it its values cause if any one will use the reference variable to make anything to this object it becomes ready and existed .

- Most of constructor using cases it to use it to initialize the state of an object , to make and assign values to the object instance variables and we can make overloaded constructors , the non arg constructor is the same as the default constructor one
- If you write a constructor that takes arguments and you still want a no-arg constructor, you'll have to build the no-arg constructor yourself!
- If you write a constructor that takes arguments and you still want a no-arg constructor, you'll have to build the no-arg constructor yourself! And this is the role of the overloaded concept for the methods and for the special methods too so be careful about that .
- As we know in the previous chapter that if there is class x extends class y and you create an object from class x so that means you will find in the heap just one object which is the object from class x but inside it you will fins space for storing the x object instance variables itself and space for the class y inherited instance variables too
- This is role : all the constructors in an object inheritance tree must be run when you create a new object
- For an object to be fully formed, all the superclass parts of itself must be fully formed, and that’s why the superclass constructor must run. All instance variables from every class in the inheritance tree have to be declared and initialized. And this is called CONSTRUCTOR CHAINING
- The only way to call a superclass constructor is by calling super() and if you do not call the super () the compiler will call It automatically and the compiler will always call the super() with no arguments that will call the super class default constructor .
- The super () call if you will put it you must put it the first statement in the child constructor or the compiler will automatically do that .
- We will take now the keyword THIS which we will treat it as a reference to the current object okay now as we know that in java each class can have multiple overloaded constructors and that may lead you to write duplication code in each constructor so if there is change happened you will need to change all the constructors codes so what about if we make the whole code in just one

place and the another constructors call each of them inside them until reach to this original code? So this will solve the duplication code problem and if there is change happened we can solve it easily in just one place but how we can make that? This will be happened by using the THIS keyword it is using to call the constructors but if the same class but the super is used to call the constructors but of the parent class and both must be in the first statement in the constructor so that means both can not be in the same constructor cause the compiler need to know If I will go to finish my initialization first or going to finishing my parent variables first? So each constructor will have either this or super and this is an example for this :

```
public Mini() {
  this(Color.RED);
}

public Mini(Color c) {
  super("Mini");
  color = c;
}
```

Now if there is change happened we will just need to change the code if the second constructor and the first one will still the same

- A local variable lives only within the method that declared the variable but An instance variable lives as long as the object does. If the object is still alive, so are its instance variables.
- An object is alive as long as there are live references to it. If a reference variable goes out of scope but is still alive, the object it refers to 1s still alive on the Heap. And then you have to ask…"“What happens when the Stack frame holding the reference gets popped off the Stack at the end of the method?” If that was the only live reference to the object, the object is now abandoned on the Heap. The reference variable disintegrated with the Stack frame, so the abandoned object is now, officially, toast. The trick is to know the point at which an object becomes eligible for garbage collection.