Lecture 5: Signature of main method, Garbage collection

Explanation of the signature of main method:

"public static void main (string…args) or (string args[])"

- public -> The main method must be public because it is call from outside the class . It is called by JVM.
- static -> the main method must be static so that without creating an instance JVM can call it .

If the main method were not declared static then JVM has to create instance of main class.

- void -> Main method doesn't returns anything therefore it must be void .
- main ->Entry point of a program.
- String args [ ] -> It is used to receive any arbitrary number of arguments and save it in an array .

Meaning of each token of "System.out.println()":

System is a predefined class that provides access to the system.

Out is a static field.

Println() or print() is used to display string which is passed to it. It is a method of PrintStream class.

Garbage collection:

- The **finalize()** method is equivalent to a **destructor** of C++.
- When the job of an object is over, or to say, the object is no more used in the program, the object is known as **garbage**.
- The process of removing the object from a running program is known as **garbage collection**.
- The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects.
- Before an object is garbage collected, the JRE (Java Runtime Environment) calls the finalize() method.
- finalize() method can be best utilized by the programmer to close the I/O streams, JDBC connections or socket handles etc.
- Although **C++ destructor** and a **Java finalize()** methods similar in many respects, their actual operation is distinctively different.
- **In C++,** when an object goes out of scope, it is destroyed. Just prior to its destruction, its destructor function is called (if it has one). This is a hard-and-fast rule. There are no exceptions. The rule is following
- **Every object is destroyed when it goes out of scope:** *Thus, if we declare a local object inside a function, when that function returns, that local object is automatically destroyed. The same goes for function parameters and for objects returned by functions.*
- **Just before destruction, the object's destructor is called:** *This happens immediately, and before any other program statements will execute. Thus, a C++ destructor will always execute in a deterministic fashion. We can always know when and where a destructor will be executed.*

- **In Java,** the tight linkage of the destruction of an object and the calling of its **finalize()** method does not exist. In Java, objects are not explicitly destroyed when they go out of scope. Rather, an object is marked as unused when there are no longer any references pointing to it. Even then, the **finalize()** method will not be called until the garbage collector runs. Thus, we cannot know precisely when or where a call to **finalize()** will occur. Even if we execute a call to **gc()** (the garbage collector), there is no guarantee that **finalize()** will immediately be executed.
- Calling the gc method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all discarded objects.
- **E.g.**

```
class Garbage{
	long i;
	Garbage(long i){
		this.i=i;
	}
	public void finalize()  {
		System.out.println("FINALISING: "+ i);
	}
	static public void main(String[] a){
		for(long i=0; i<10; i++){
			System.out.println("H");
			new Garbage(i);
				if(i==3)System.gc();
		}
	}
}
```