Creating thread by Inheriting Thread class (Class Based)

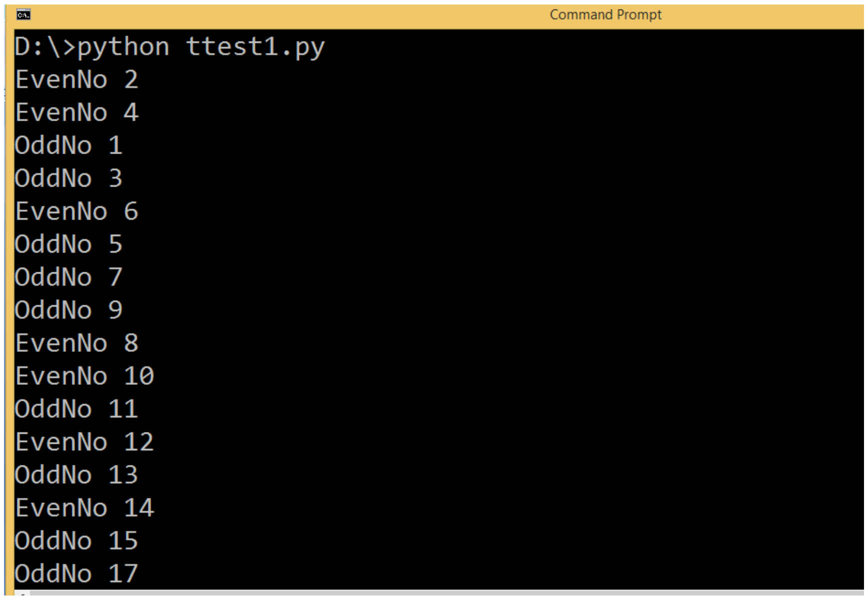
1. Develop a class by inheriting Thread class
2. Write a constructor which calls the constructor of super class (Thread class)
3. Override run method of Thread class (run method provides operation performed by thread)
4. Create object of user defined thread class
5. Execute thread by invoking start() method

Example:

```
import threading

class EvenThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        for num in range(1,21):
            if num%2==0:
                print(f'EvenNo {num}')

class OddThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        for num in range(1,21):
            if num%2!=0:
                print(f'OddNo {num}')



t1=EvenThread()
```

```
t2=OddThread()
t1.start()
t2.start()
```

Output


Command Prompt

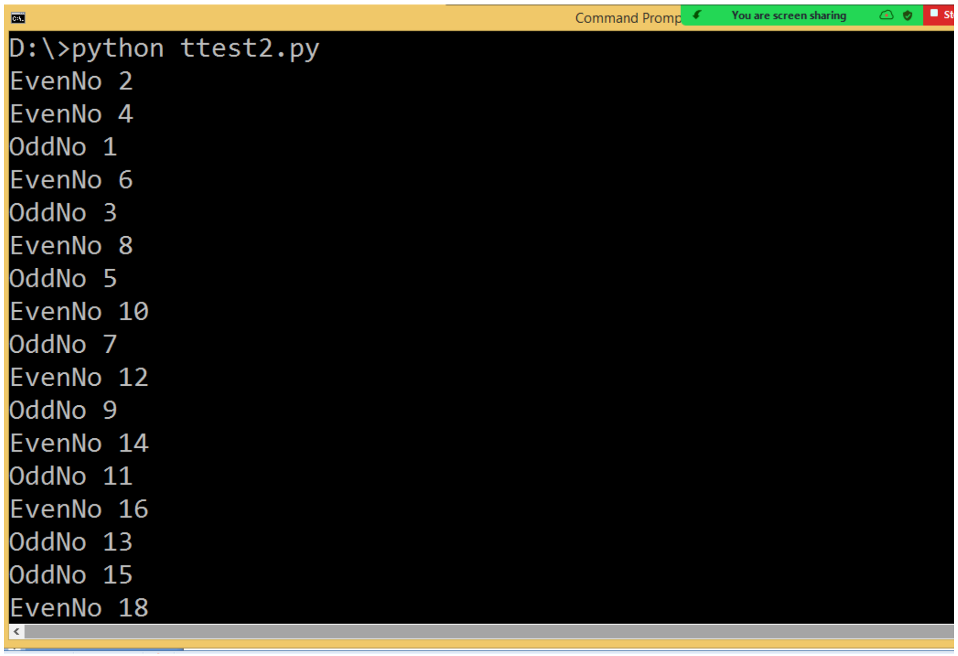
```
D:\>python ttest1.py
EvenNo 2
EvenNo 4
OddNo 1
OddNo 3
EvenNo 6
OddNo 5
OddNo 7
OddNo 9
EvenNo 8
EvenNo 10
OddNo 11
EvenNo 12
OddNo 13
EvenNo 14
OddNo 15
OddNo 17
```

Example:

```
import threading

def even(start,stop):
    for num in range(start,stop+1):
        if num%2==0:
            print(f'EvenNo {num}')

def odd(start,stop):
    for num in range(start,stop+1):
        if num%2!=0:
            print(f'OddNo {num}')
```

```
t1=threading.Thread(target=even,args=(1,20))
t2=threading.Thread(target=odd,kwargs={'start':1,'stop':20})
t1.start()
t2.start()
```

Output



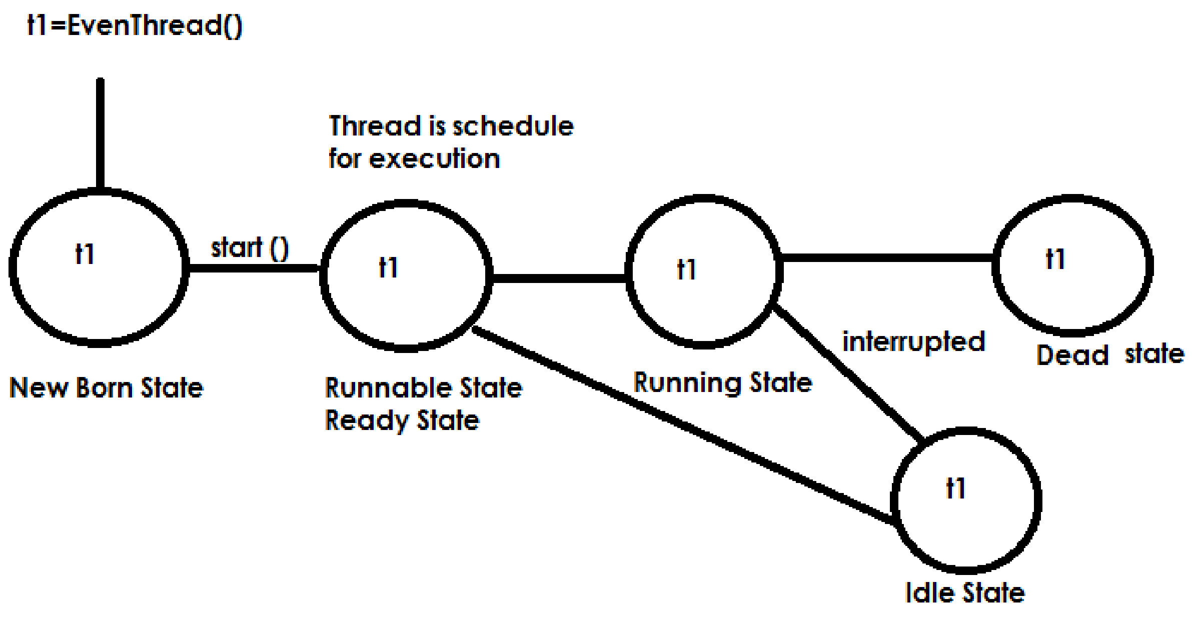
```
D:\>python ttest2.py
EvenNo 2
EvenNo 4
OddNo 1
EvenNo 6
OddNo 3
EvenNo 8
OddNo 5
EvenNo 10
OddNo 7
EvenNo 12
OddNo 9
EvenNo 14
OddNo 11
EvenNo 16
OddNo 13
OddNo 15
EvenNo 18
```

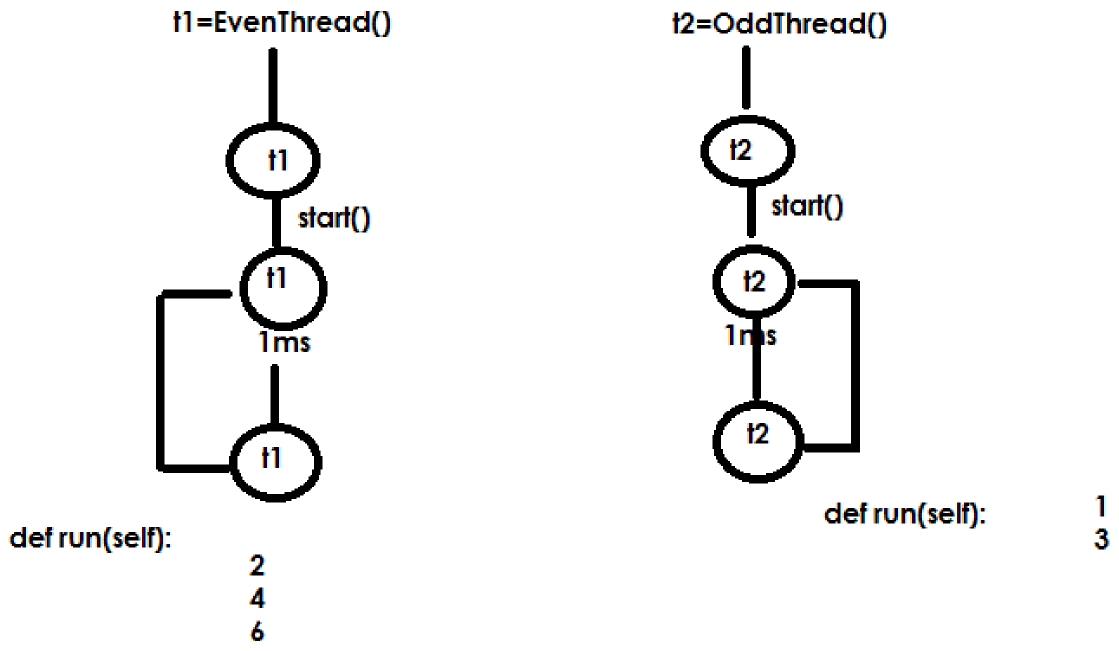
Life Cycle of Thread

Thread execution is schedule by thread scheduler provide by PVM (Python Virtual Machine).

States of Thread

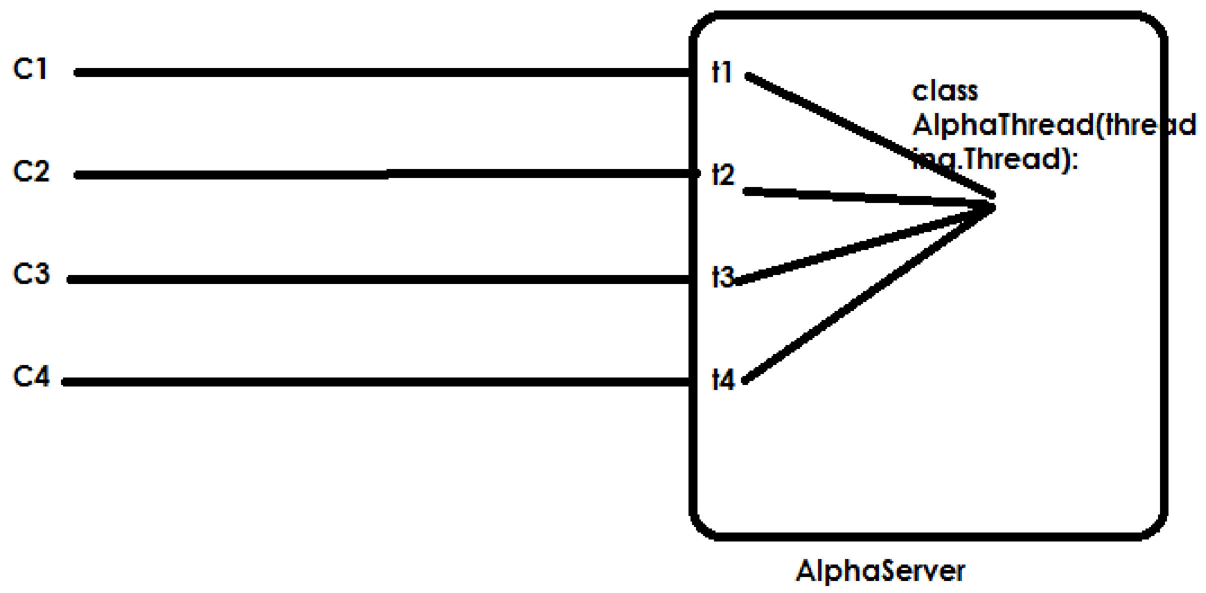
1. New Born State
2. Runnable State
3. Running State
4. Idle State
5. Dead State





name

A string used for identification purposes only. It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.



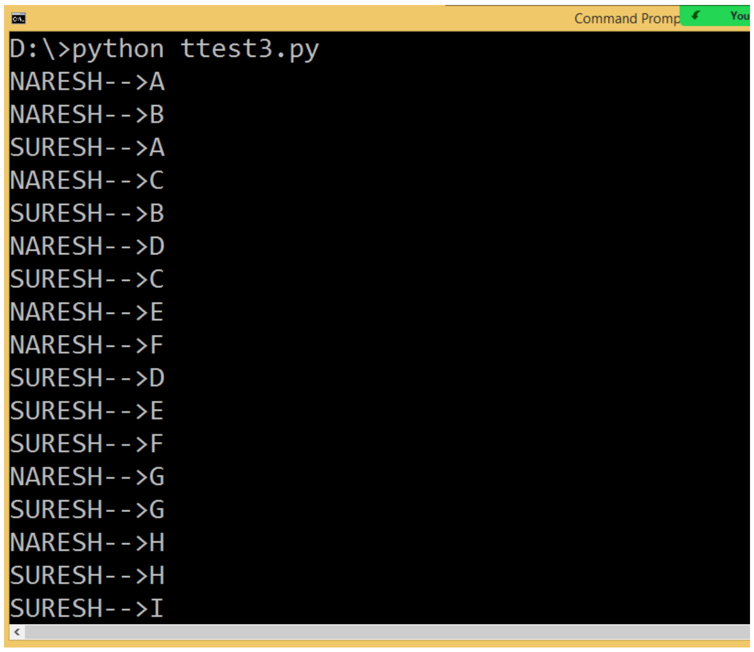
Example:

```
import threading

class AlphaThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        for n in range(65,91):
            print(f'{super().name}-->{chr(n)}')
```

```
# AlphaServer
t1=AlphaThread()
t2=AlphaThread()
t1.name="NARESH"
t2.name="SURESH"
```

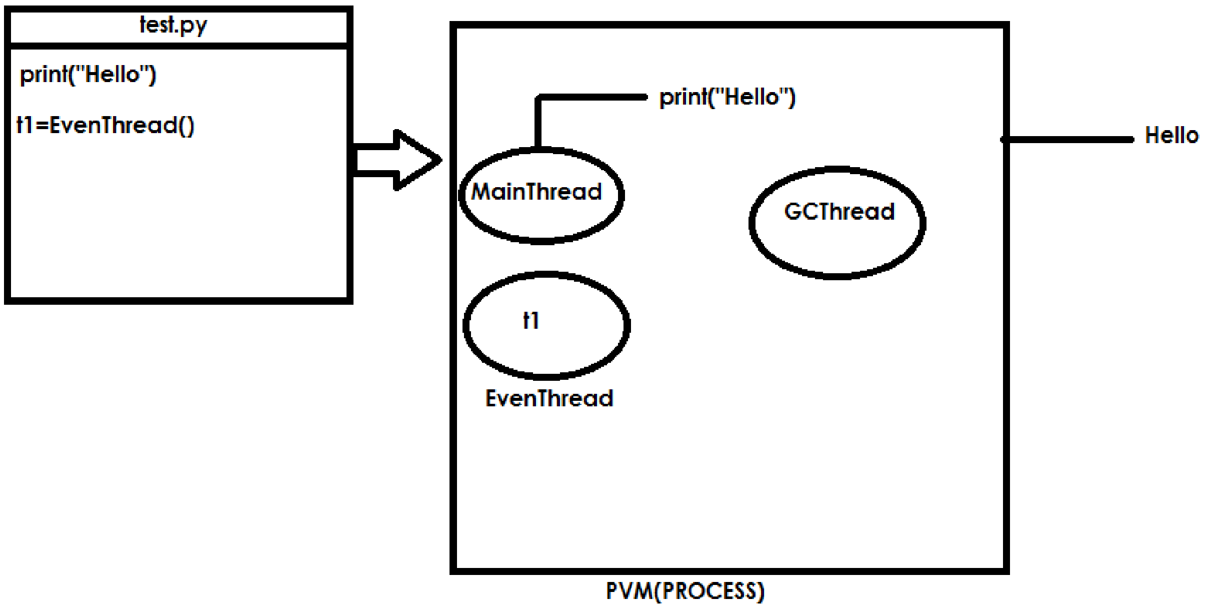
```
t1.start()
t2.start()
```

Output



```
D:\>python ttest3.py
NARESH-->A
NARESH-->B
SURESH-->A
NARESH-->C
SURESH-->B
NARESH-->D
SURESH-->C
NARESH-->E
NARESH-->F
SURESH-->D
SURESH-->E
SURESH-->F
NARESH-->G
SURESH-->G
NARESH-->H
SURESH-->H
SURESH-->I
```

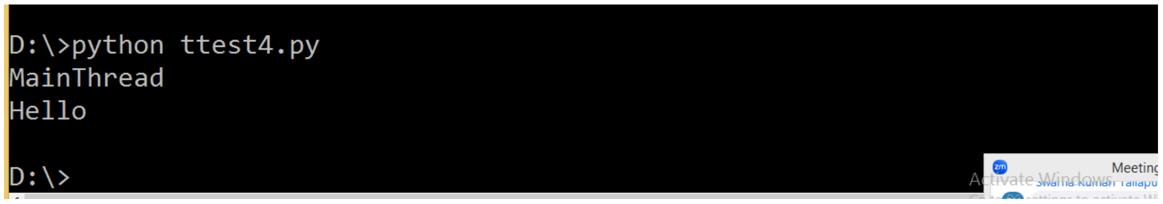
Python is multithreaded, every python program is executed inside PVM process as a thread. The default thread created by PVM is MainThread.



Example:

```
import threading

t1=threading.current_thread()
print(t1.name)
print("Hello")
```

Output



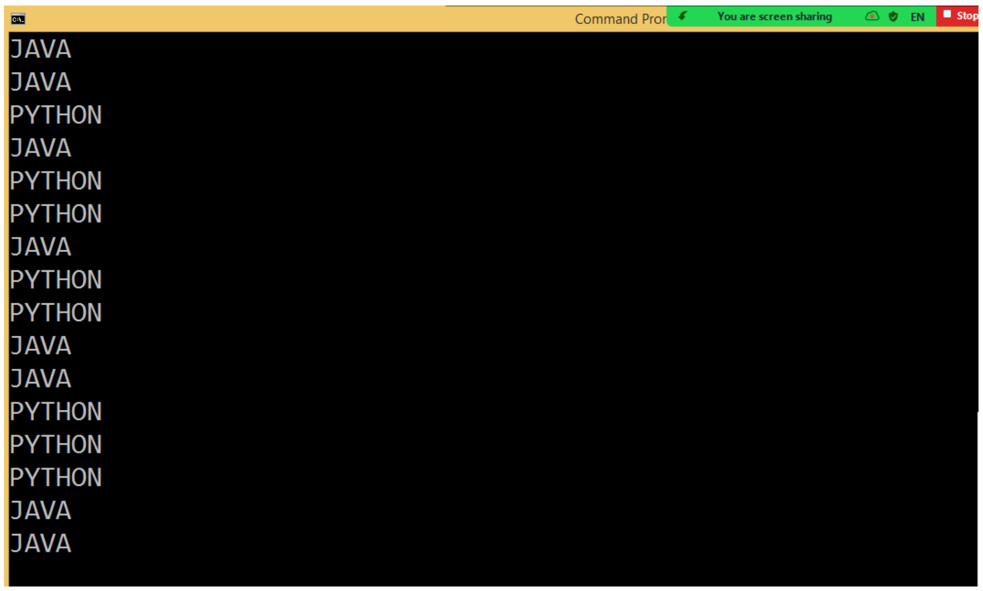
```
D:\>python ttest4.py
MainThread
Hello

D:\>
```

Example:

```
import threading
def print_msg():
    for i in range(10):
        print("PYTHON")
```

```
# MainThread
t1=threading.Thread(target=print_msg)
t1.start()
for j in range(10):
    print("JAVA")
```

output



```
JAVA
JAVA
PYTHON
JAVA
PYTHON
PYTHON
JAVA
PYTHON
PYTHON
JAVA
JAVA
PYTHON
PYTHON
PYTHON
JAVA
JAVA
```

join(timeout=None)

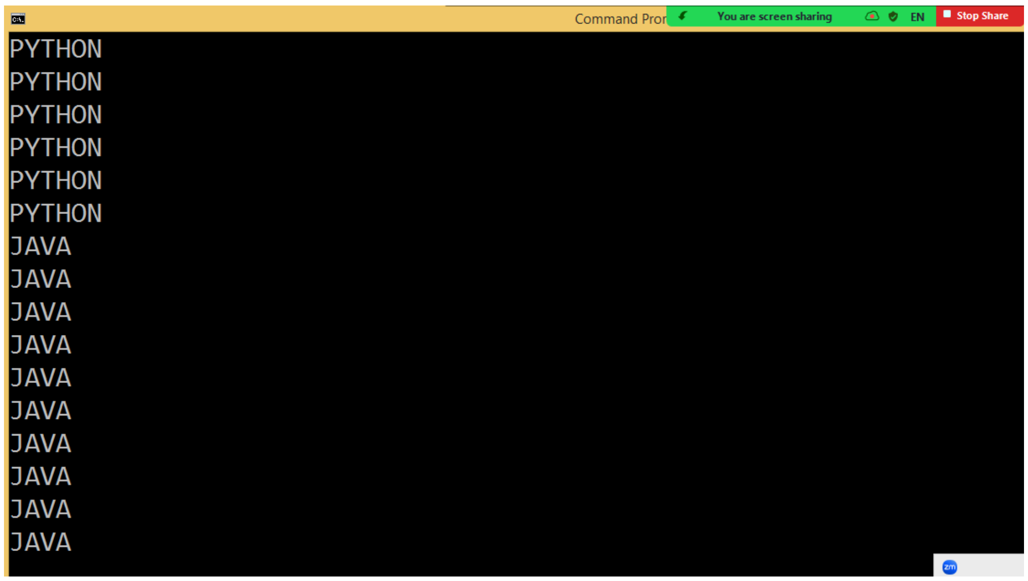
Wait until the thread terminates. This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception – or until the optional timeout occurs.

Example: if thread1 calls join method of thread-2, thread-1 waits until execution of thread2 is completed or terminated.

Example

```
import threading
def print_msg():
    for i in range(10):
        print("PYTHON")


# MainThread
t1=threading.Thread(target=print_msg)
t1.start()
t1.join()
for j in range(10):
    print("JAVA")
```

Output

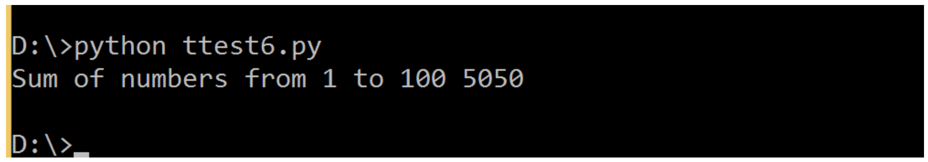


```
PYTHON
PYTHON
PYTHON
PYTHON
PYTHON
PYTHON
JAVA
JAVA
JAVA
JAVA
JAVA
JAVA
JAVA
JAVA
JAVA
JAVA
```

Example

```
import threading

s=0
def sum_of_numbers():
    global s
    for num in range(1,101):
      s=s+num



# MainThread
t1=threading.Thread(target=sum_of_numbers)
t1.start()
t1.join()
print(f'Sum of numbers from 1 to 100 {s}')
```

Output



```
D:\>python ttest6.py
Sum of numbers from 1 to 100 5050

D:\>
```