Thread class Constructor :
---
* Thread class has provided total 10 constructors. The following are the commonly used constructor in the Thread class.

1) Thread t1 = new Thread();

2) Thread t2 = new Thread(String name);

3) Thread t3 = new Thread(Runnable target); //Loose Coupling (We can assign any class object which implements from Runnable interface)

4) Thread t4 = new Thread(Runnable target, String name);

5) Thread t5 = new Thread(ThreadGroup group, String name);

6) Thread t6 = new Thread(ThreadGroup group, Runnable target);

7) Thread t7 = new Thread(ThreadGroup group, Runnable target, String name);

Runnable interface by using different Cases :

Case 1 :

Runnable interface by using Anonymous inner class :

```
package com.ravi.runnable_cases;

public class RunnableCase1
{
    public static void main(String[] args)
    {
        //Anonymous Inner class
        Runnable r1 = new Runnable()
        {
            @Override
            public void run()
            {
                String name = Thread.currentThread().getName();
                System.out.println("Thread Name is :"+name);
            }
        };

        Thread thread = new Thread(r1);
        thread.start();

    }

}
```

Case 2 :
---
* Implementation of Runnable by using Anonymous inner class without ref. using Thread class cons :

```
package com.ravi.runnable_cases;

public class RunnableCase2
{
    public static void main(String[] args)
    {
        Thread t1 = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                String name = Thread.currentThread().getName();
                System.out.println("Thread Name is :"+name);
            }
        });

        t1.start();
    }
}
```

Case 3 :
---
* Implemntation of Runnable by using Lambda expression

```
package com.ravi.runnable_cases;

public class RunnableCase3
{
    public static void main(String[] args)
    {
        Runnable r1 = () ->
        {
            String name = Thread.currentThread().getName();
            System.out.println("Thread Name is :"+name);
        };

        Thread t1 = new Thread(r1);
        t1.start();
    }
}
```

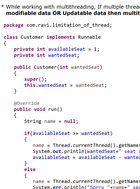
Case 4 :

```
package com.ravi.runnable_cases;

public class RunnableCase4
{
    public static void main(String[] args)
    {

        Thread t1 = new Thread(()-> System.out.println(Thread.currentThread().getName()));
        t1.start();

        System.out.println("......................");

        new Thread(()-> System.out.println(Thread.currentThread().getName()),"Child5").start();

    }

}
```

Limitation of Multithreading :
---
* Multithreading is very good to complete our task as soon as possible due to concurrent execution.

* In some situation, Multithreading may produce wrong result OR wrong data, this situation is known as "Data Race OR race condition".

Example 1:



* While working with multithreading, If multiple threads are **accessing the data and if the data is modifiable data OR Updatable data then multithreading may produce wrong result.**

```
package com.ravi.limitation_of_thread;

class Customer implements Runnable
{
    private int availableSeat = 1;
    private int wantedSeat;

    public Customer(int wantedSeat)
    {
        super();
        this.wantedSeat = wantedSeat;
    }

    @Override
    public void run()
    {
        String name = null;

        if(availableSeat >= wantedSeat)
        {
            name = Thread.currentThread().getName();
            System.out.println(wantedSeat+ " seat is reserved for :"+name);
            availableSeat = availableSeat - wantedSeat;
        }
        else
        {
            name = Thread.currentThread().getName();
            System.err.println("Sorry "+name+" seat is not available..");
        }
    }
}


public class RailwayReservation
{
    public static void main(String[] args) throws InterruptedException
    {
        Customer c1 = new Customer(1);

        Thread t1 = new Thread(c1, "Smith");
        Thread t2 = new Thread(c1, "Alen");

        t1.start();
        t2.start();

    }
}
```

Note : In this program, smith and Alen both will get the ticket.

```
package com.ravi.limitation_of_thread;

class Customer
{
    private double balance = 20000;
    private double withAmount;

    public Customer(double withAmount)
    {
        super();
        this.withAmount = withAmount;
    }

    public void withdraw()
    {
        String name = null;

        if(this.withAmount <= this.balance)
        {
            name = Thread.currentThread().getName();
            System.out.println(this.withAmount+" amount is withdraw by :"+name);
            this.balance = this.balance - this.withAmount;
            System.out.println("Current Balance is :"+this.balance);
        }
        else
        {
            name = Thread.currentThread().getName();
            System.err.println("Sorry!!"+name+ " you have insufficient Balance");
        }

    }
}

public class BankingApplication {

    public static void main(String[] args) throws InterruptedException
    {
        Customer cust = new Customer(20000);

        Runnable r1 = () -> cust.withdraw();


        Thread t1 = new Thread(r1, "Scott");
        Thread t2 = new Thread(r1, "Smith");

        t1.start();
        t2.start();

    }

}
```

Note : Scott and Smith both will get 20000 amount.
