Java Assignment-3

1. Differences between Runnable Interface and Extends in Threads concept.

Runnable Interface	Extends
Implementing the Runnable interface gives you the choice to extend any class you like , but still define behavior that will be run by separate thread.	If you extend Thread, you can not extend anything else . Java does not support multiple inheritance. In reality , you do not need Thread class behavior , because in order to use a thread you need to instantiate one anyway.
In this, creating a different Runnable class for a specific behavior job (if the work you want to be done is *job*). It gives us the freedom to reuse the specific behavior job whenever required.	It contains both thread and job specific behavior code. Hence once thread completes execution , it can not be restart again.
It makes the code loosely-coupled and easier to read .Because the code is split into two classes . Thread class for the thread specific code and your Runnable implementation class for your job that should be run by a thread code.	It makes the code tightly coupled . Single class contains the thread code as well as the job that needs to be done by the thread.
A class can implement one or more interfaces.	A class can extend one super class.
An interface cannot implement other interface	An interface can extend one or more interfaces

2. Producer-Consumer Problem using Threads

```
class Q
{
        int n;
        boolean statusFlag = false;
        synchronized void put(int n)
        {
```

```
		try
		{
			while(statusFlag)
			{
				wait();
			}
		}
		catch(InterruptedException e){}
		this.n = n;
		System.out.println("Put: " + n);
		statusFlag = true;
		notify();
	}
	synchronized int get()
	{
		try
		{
			while(!statusFlag)
			{
				wait();
			}
		}
		catch(InterruptedException e){}
		statusFlag = false;
		System.out.println("Got: " + n);
		notify();
		return n;
	}
}

class Prod implements Runnable
{
	Q q;
	Prod(Q q)
	{
		this.q = q;
		new Thread(this, "Producer").start();
	}
	public void run()
	{
		int i = 0;
		while(i<=10)
		{
			q.put(i++);
		}
	}
}
```

```
class Consumer implements Runnable
{
	Q q;
	Consumer(Q q)
	{
		this.q = q;
		new Thread(this, "Consumer").start();
	}
	public void run()
	{
		while(true)
		{
			q.get();
		}
	}
}
public class Prodcons
{
	public static void main(String[] args)
	{
		Q q = new Q();
		Prod p = new Prod(q);
		Consumer c = new Consumer(q);
	}
}
```

Output:

```
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
Got: 5
Put: 6
Got: 6
Put: 7
Got: 7
Put: 8
Got: 8
Put: 9
Got: 9
```

Put: 10
Got: 10