Course code : **CSE2005**

Course title : **Object Oriented Programming**

Creating Threads

Objectives

This session will give the knowledge about

- Creating Threads By extending Thread Class
- Creating Threads By implementing Runnable interface

Extending Thread

We can also create Threads by extending the Thread class:

- Instantiate the class that extends Thread
- This class must override run() method
- The code that should run as a thread will be part of this run() method
- We must call the start() method on this thread
- start( ) in turn calls the thread’s run( ) method

Extending Thread Example

```
public class ThreadDemo extends Thread {
        public void run() {
                System.out.println("thread is running...");
        }
        public static void main(String args[]) {
                ThreadDemo t1=new ThreadDemo();
                t1.start();
        }
}
```

Example-2

```
public class ThreadDemo extends Thread {
        public void run() {
                for(int counter=1;counter<=100;counter++){
                        System.out.println("Thread is running "+counter);
                }
        }
        public static void main(String args[]) {
                ThreadDemo t1=new ThreadDemo();
                t1.start();
        }
}
```

The main Thread

When a Java program starts executing:

- the main thread begins running
- the main thread is immediately created when main() commences execution

Information about the main or any thread can be accessed by obtaining a reference to the thread using a public, static method in the Thread class called currentThread()

Obtaining Thread-Specific Information

```
class MyThread extends Thread {
        public void run() {
                for (int i = 0; i < 3; i++)
                        System.out.println(Thread.currentThread().getName());
        }
}
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread t1=new MyThread();
                t1.setName("college");
```

Obtaining Thread-Specific Information

```
                    t1.start();
                    Thread t2=Thread.currentThread();
                    t2.setName("company");
                    for (int i = 0; i < 3; i++)
                            System.out.println(Thread.currentThread().getName());
            }
}
```

currentThread()

```
public class ThreadDemo {
        public static void main(String args[]) {
                Thread t1=Thread.currentThread();
            System.out.println(t1);
        }
}
```

```
Thread[main,5,main]
```

Because thread.toString() returns a string representation of this thread, including the thread's name, priority, and thread group.

Course code : **CSE2005**

Course title : **Object Oriented Programming**

Creating Threads By implementing Runnable interface

Creating Threads: Implementing Runnable

Thread can be created by creating a class which implements Runnable interface.

```
public class ThreadDemo implements Runnable {
        @Override
        public void run() {
        }
        public static void main(String args[]) {
        }
}
```

Creating Threads: Implementing Runnable

- After defining the class that implements Runnable, we have to create an object of type Thread from within the object of that class. This thread will end when run( ) returns or terminates.
- This is mandatory because a thread object confers multithreaded functionality to the object from which it is created.
- Therefore, at the moment of thread creation, the thread object must know the reference of the object to which it has to confer multithreaded functionality. This point is borne out by one of the constructors of the Thread class.

Creating Threads: Implementing Runnable

- The Thread class defines several constructors one of which is:
    - **Thread(Runnable threadOb, String threadName)**
- In this constructor, “threadOb” is an instance of a class implementing the Runnable interface and it ensures that the thread is associated with the run( ) method of the object implementing Runnable
- When the run() method is called, the thread is believed to be in execution.
- String threadName – we associate a name with this thread.

Creating Threads: Implementing Runnable

- Creating a thread does not mean that it will automatically start executing.
- A thread will not start running, until you call its **start( )** method.
- The **start( )** method in turn initiates a call to **run( ).**

Creating Threads: Implementing Runnable

```
public class ThreadDemo implements Runnable {
        public void run() {
                for (int i = 0; i < 3; i++)
                        System.out.println("thread is running...");
        }
        public static void main(String args[]) {
                ThreadDemo obj = new ThreadDemo();
                Thread t1 = new Thread(obj);
                t1.start();
        }
}
```

Implementing Runnable or Extending Thread ?

- When we extend Thread class, we can’t extend any other class even we require and When we implement Runnable, we can save a space for our class to extend any other class in future or now.
- When we extend Thread class, each of our thread creates unique object and associate with it. When we implements Runnable, it shares the same object to multiple threads.
- So, if the sole aim is to define an entry point for the thread by overriding the run( ) method and not override any of the Thread class’ other methods, it is recommended to implement the Runnable interface

Creating Multiple threads

You can launch as many threads as your program needs The following example is a program spawning multiple threads

```
public class ThreadDemo implements Runnable {
        public void run() {
                for (int i = 0; i < 3; i++)
                        System.out.println(Thread.currentThread());
        }
        public static void main(String args[]) {
                ThreadDemo obj = new ThreadDemo();
                Thread t1 = new Thread(obj);
```

Creating Multiple threads

```
                    t1.setName("first");
                    t1.start();
                    Thread t2 = new Thread(obj);
                    t2.setName("second");
                    t2.start();
          }
}
```

How Thread works?

```
class MyThread1 extends Thread {
        public void run() {
                try {
                        for (int i = 0; i < 5; i++) {
                        System.out.println(Thread.currentThread().getName() + i);
                                Thread.sleep(1000);
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

How Thread works?

```
class MyThread2 extends Thread {
        public void run() {
                System.out.println(Thread.currentThread().getName());
                Scanner sin = new Scanner(System.in);
                int id = sin.nextInt();
                System.out.println(id);
        }
}
```

How Thread works?

```
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread1 t1 = new MyThread1();
                t1.setName("first");
                MyThread2 t2 = new MyThread2();
                t2.setName("second");
                t1.start();
                t2.start();
        }
}
```

Summary

We have discussed about

- Creating Threads By extending Thread Class
- Creating Threads By implementing Runnable interface