Multithreading

Creating Threads By Extending Thread Class

Agenda

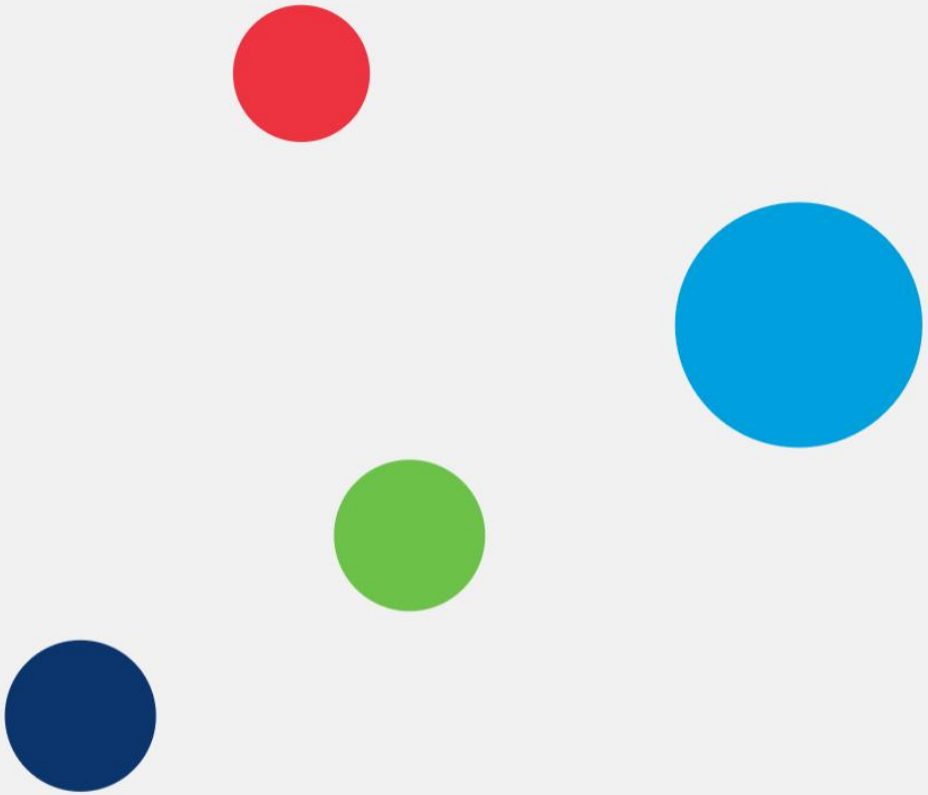
1 **Creating Threads By Extending Thread Class**

Objectives

At the end of this module, you will be able to:

- How to create threads by extending Thread class

Creating Threads By Extending Thread Class



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

A very simple demo for creating threads by extending Thread class:-

```
public class ThreadDemo1 extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
ThreadDemo1 threadDemo=new ThreadDemo1();
threadDemo.start();
 }
}
```

Extending Thread Example (Contd.).

One More Demo to show that thread is Running:-

```
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 threadDemo=new ThreadDemo();
threadDemo.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

```
public class ThreadInfo {
  public static void main(String args[]) {
    Thread t = Thread.currentThread( );
    System.out.println("Current Thread :" + t);
    t.setName("Demo Thread");
    System.out.println("New name of the thread :"  +  t);
    try {
        Thread.sleep(1000);
    }
    catch (InterruptedException e) {
        System.out.println("Main Thread Interrupted");
    }
  }
}
```

Obtaining Thread-Specific Information (Contd.).

```
public static void main(String args[]) {
  Thread t = Thread.currentThread( );
  System.out.println("Current Thread :" + t);
  t.setName("Demo Thread");
  System.out.println("New name of the thread :"  +  t);
  try {
      Thread.sleep(1000);
  }
  catch (InterruptedException e) {
      System.out.println("Main Thread Interrupted");
  }
} }
```

Assignment

Summary

- Creating threads by extending Thread class

Thank You

Multithreading

Creating Threads with Runnable Interface

Agenda

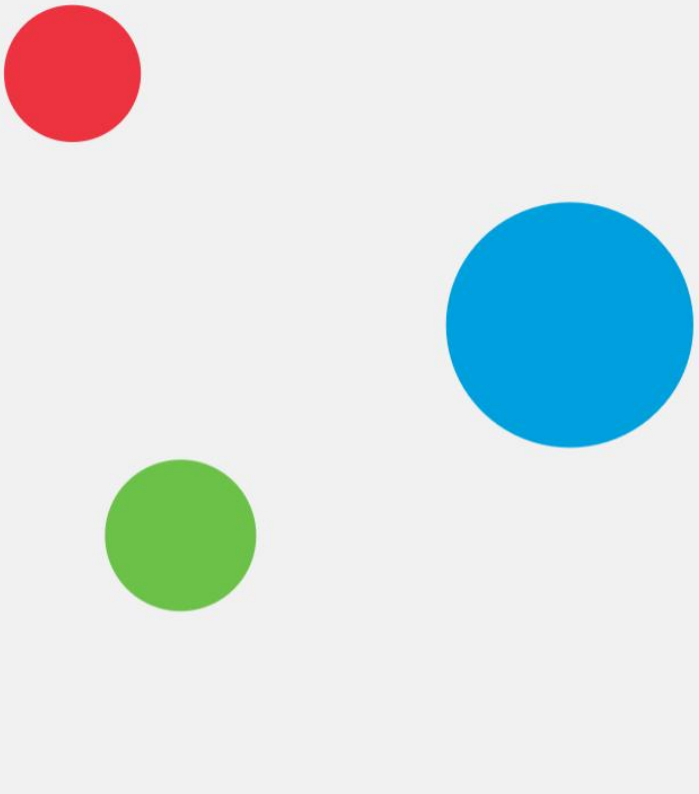
1. Creating Threads with Runnable Interface
2. Creating Multiple threads

Objectives

At the end of this module, you will be able to:

- How to create threads by implementing Runnable interface

Creating Threads with Runnable Interface



Creating Threads: Implementing Runnable

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

```
class DemoThread implements 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 (Contd.).

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 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 (Contd.).

A very simple demo for creating threads by implementing Runnable Interface:-

```
public class ThreadDemo implements Runnable {

public void run() {
System.out.println("thread is running...");
}

public static void main(String args[]) {
ThreadDemo threadDemo = new ThreadDemo();
Thread t1 = new Thread(threadDemo);
t1.start();
}
}
```

Output: thread is running…

Creating Threads: Implementing Runnable (Contd.).

One More Demo to show that thread is Running:-

```
public class ThreadDemo implements Runnable {

public void run() {
for(int counter=1;counter<=100;counter++){
System.out.println("thread is running..."+counter);
}
}

public static void main(String args[]) {
ThreadDemo threadDemo = new ThreadDemo();
Thread t1 = new Thread(threadDemo);
t1.start();
}

}
```

Output

thread is running...1

thread is running...2

thread is running...3

thread is running...4

thread is running...5

thread is running...6

thread is running...7

thread is running...8

…

thread is running...89

thread is running...90

thread is running...91

thread is running...92

thread is running...93

thread is running...94

thread is running...95

thread is running...96

thread is running...97

thread is running...98

thread is running...99

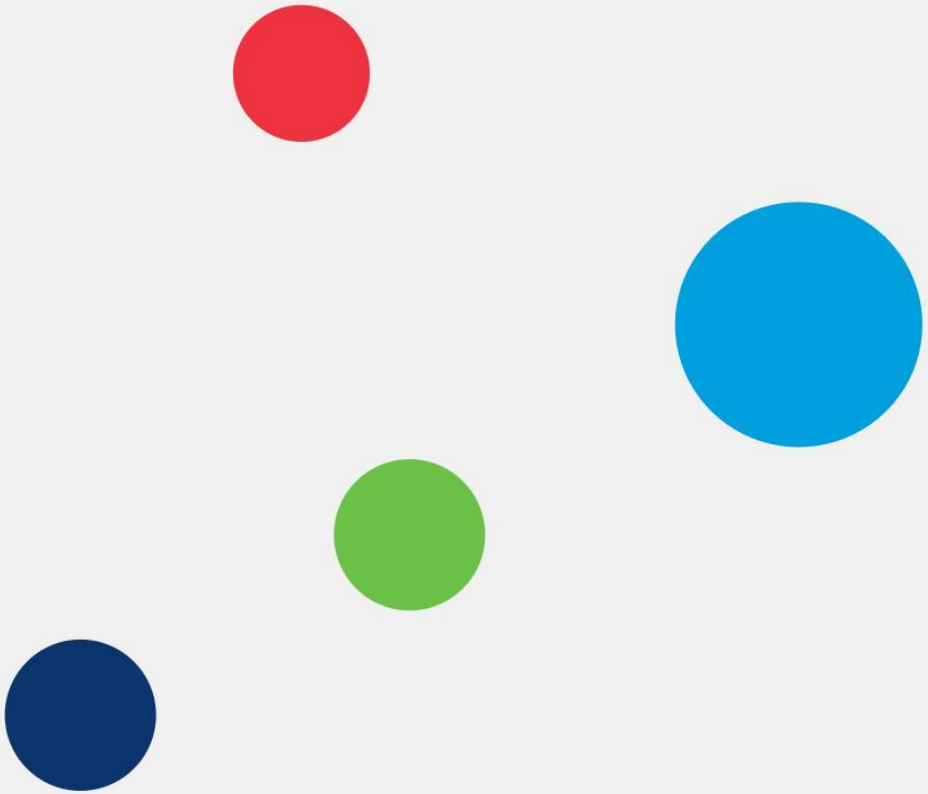
thread is running...100

Implementing Runnable or Extending Thread ?

- A modeling heuristic pertaining to hierarchies says that classes should be extended only when they are enhanced or modified in some way

- 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



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 counter=1;counter<=100;counter++){
System.out.println(Thread.currentThread().getName()+"thread is
running..."+counter);
}
}
```

Creating Multiple Threads (Contd.).

```
public static void main(String args[]) {
ThreadDemo threadDemo = new ThreadDemo();
Thread t1 = new Thread(threadDemo,"First");
Thread t2 = new Thread(threadDemo,"Second");
t1.start();
t2.start();
}

}
```

Output

Firstthread is running...1

Secondthread is running...1

Firstthread is running...2

Secondthread is running...2

Secondthread is running...3

Secondthread is running...4

Secondthread is running...5

Secondthread is running...6

Secondthread is running...7

Secondthread is running...8

Secondthread is running...9

Secondthread is running...10

Secondthread is running...11

Secondthread is running...12

Secondthread is running...13

Secondthread is running...14

Secondthread is running...15

Secondthread is running...16

Secondthread is running...17

Secondthread is running...18

…

Time to think…!!!

- In the program that was just demonstrated, if we replace t.start() in the constructor of DemoThread with t.run(), what will be the consequence ?
- Will all the threads run? Beginning with the commencement of main() and till the end of main(), how many threads in total get executed?

1. *No difference. The program runs successfully with four threads(3 kid threads and main), running simultaneously as the program that was just demonstrated*
2. *The program runs successfully, but all the four threads run one after the other*
3. *The program fails to compile*
4. *The program executes without any error but only one thread gets executed, i.e. main() thread*
5. *The program throws "Thread not started exception"*

Summary

- Creating threads by implementing Runnable Interface
- Creating multiple threads

Thank You