Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 1

MULTITHREADING AND GENERIC PROGRAMMING

4.3 Creating Threads

COMPUTER SCIENCE & ENGINEERING

CREATING THREADS

Introduction

- Threading is a facility to allow multiple tasks to run concurrently within a single process. Threads are independent, concurrent execution through a program, and each thread has its own stack.
- In Java, There are two ways to create a thread:
  - By extending Thread class.
  - By implementing Runnable interface.

CREATING THREADS

Java Thread Benefits

- .Java Threads are lightweight compared to processes as they take less time and resources to create a thread.
- Threads share their parent process data and code
- Context switching between threads is usually less expensive than between processes.
- Thread intercommunication is relatively easier than process communication.
- thread class:Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

CREATING THREADS

Commonly used Constructors of thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

CREATING THREADS

Commonly used methods of thread class:

1. public void run(): is used to perform action for a thread.
2. public void start(): starts the execution of the thread. JVM calls the run() method on the thread.
3. public void sleep(long milliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. public void join(): waits for a thread to die.
5. public void join(long milliseconds): waits for a thread to die for the specified mili-seconds.
6. public int getPriority(): returns the priority of the thread.
7. public int setPriority(int priority): changes the priority of the thread.
8. public string getname(): returns the name of the thread.
9. public void setname(string name): changes the name of the thread.
10. public thread currentthread():returns the reference of currently executing thread.
11. public int getid(): returns the id of the thread.

CREATING THREADS

Commonly used methods of thread class:

11.public int getid(): returns the id of the thread.

12.public thread.state getstate(): returns the state of the thread.

13.public boolean isalive(): tests if the thread is alive.

14.public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

15.public void suspend(): is used to suspend the thread(depricated).

16.public void resume(): is used to resume the suspended thread(depricated).

17.public void stop(): is used to stop the thread(depricated).

18.public boolean isdaemon(): tests if the thread is a daemon thread.

19.public void setdaemon(boolean b): marks the thread as daemon or user thread.

20.public void interrupt(): interrupts the thread.

21.public boolean isinterrupted(): tests if the thread has been interrupted.

22.public static boolean interrupted(): tests if the current thread has been interrupted.

CREATING THREADS

Naming thread:

- The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on.
- But we can change the name of the thread by using setName() method.

The syntax of setName() and getName() methods are:

public string getname(): is used to return the name of a thread.

public void setname(string name): is used to change the name of a thread.

CREATING THREADS

CREATING THREAD USING EXTENDING THREAD

- The first way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
- The extending class must override the run( ) method, which is the entry point for the new thread.
- It must also call start( ) to begin execution of the new thread.

CREATING THREADS

Example : Creating a new thread by extending Thread:

```
// Create a second thread by extending Thread
class NewThread extends Thread
{
NewThread()
{ // Create a new, second thread super("Demo Thread");
System.out.println("Child thread: " + this); start(); // Start
the thread
}// This is the entry point for the second thread.
public void run()
{
try
{
for(int i = 5; i > 0; i--)
{
System.out.println("Child Thread: " + i);
Thread.sleep(500);
}
}
catch (InterruptedException e)
{
System.out.println("Child interrupted.");
}
System.out.println("Child thread is exiting");  }  }
```

```
public class ExtendThread
{
public static void main(String args[])
{
new NewThread(); // create a new thread
try
{
for(int i = 5; i > 0; i--)
{
System.out.println("Main Thread: " + i);
Thread.sleep(1000);
}
}
catch (InterruptedException e)
{
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread is exiting.");
}
}
```

Sairam

CREATING THREADS

Output: Creating a new thread by extending Thread:

(output may vary based on processor speed and task load) Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Child thread is exiting.
Main Thread: 2
Main Thread: 1
Main thread is exiting.

The child thread is created by instantiating an object of NewThread, which is derived from Thread. The call to super( ) is inside NewThread. This invokes the following form of the Thread constructor:

public Thread(String threadName)

Here, threadName specifies the name of the thread.

Sairam INSTITUTIONS

CREATING THREADS

CREATING THREAD BY IMPLEMENTING RUNNABLE

- The easiest way to create a thread is to create a class that implements the Runnable interface.
- Runnable abstracts a unit of executable code. We can construct a thread on any object that implements Runnable.
- To implement Runnable, a class need only implement a single method called run( ), which is declared as:

public void run( )

- Inside run( ), we will define the code that constitutes the new thread. The run( ) can call other methods, use other classes, and declare variables, just like the main thread can. The only difference is that run( ) establishes the entry point for another, concurrent thread of execution within the program. This thread will end when run( ) returns.

CREATING THREADS

CREATING THREAD BY IMPLEMENTING RUNNABLE contd..

- After we create a class that implements Runnable, we will instantiate an object of type Thread from within that class.
- After the new thread is created, it will not start running until we call its start( ) method, which is declared within Thread. In essence, start( ) executes a call to run( ).
- The start( ) method is shown as: void start( )

CREATING THREADS

Example: Creation of new thread by implementing Runnable:

```
// Create a second thread
class NewThread implements Runnable
{
Thread t;
NewThread()
{
// Create a new, second thread
t = new Thread(this, “Demo Thread”);
System.out.println(“Child thread: “ + t); t.start(); //
Start the thread
}
// This is the entry point for the second thread.
public void run()
{
try
{
for(int i = 5; i > 0; i--)
{
System.out.println(“Child Thread: “ + i);
Thread.sleep(500);
}
```

```
catch (InterruptedException e)
{  System.out.println(“Child interrupted.”);   }
System.out.println(“Child thread is exiting.”);
}     }
public class ThreadDemo
{
public static void main(String args[])
{
new NewThread(); // create a new thread
try
{
for(int i = 5; i > 0; i--)
{
System.out.println(“Main Thread: “ + i);
Thread.sleep(1000);
}
}
catch (InterruptedException e)
{
System.out.println(“Main thread interrupted.”);
}
System.out.println(“Main thread is exiting.”);
}
}
```

CREATING THREADS

Example: Creation of new thread by implementing Runnable:

Inside NewThread's constructor, a new Thread object is created by the following statement:

t = new Thread(this, "Demo Thread");

Passing this as the first argument indicates that we want the new thread to call the run( ) method on this object. Next, start( ) is called, which starts the thread of execution beginning at the run( ) method.

This causes the child thread's for loop to begin. After calling start( ), NewThread's constructor returns to main(). When the main thread resumes, it enters its for loop. Both threads continue running, sharing the CPU, until their loops finish.

CREATING THREADS

Output: Creation of new thread by implementing Runnable

(output may vary based on processor speed and task load) Child thread: Thread[Demo Thread,5,main]

Main Thread: 5

Child Thread: 5

Child Thread: 4

Main Thread: 4

Child Thread: 3

Child Thread: 2

Main Thread: 3

Child Thread: 1

Child thread is exiting.

Main Thread: 2

Main Thread: 1

Main thread is exiting.

CREATING THREADS

Example: Creation of new thread by implementing Runnable:

In a multithreaded program, often the main thread must be the last thread to finish run- ning. In fact, for some older JVMs, if the main thread finishes before a child thread has completed, then the Java run-time system may “hang.” The preceding program ensures that the main thread finishes last, because the main thread sleeps for 1,000 milliseconds between iterations, but the child thread sleeps for only 500 milliseconds. This causes the child thread to terminate earlier than the main thread.

Choosing an approach

The Thread class defines several methods that can be overridden by a derived class. Of these methods, the only one that must be overridden is run(). This is, of course, the same method required when we implement Runnable. Many Java programmers feel that classes should be extended only when they are being enhanced or modified in some way. So, if we will not be overriding any of Thread’s other methods, it is probably best simply to implement Runnable.

Video Link

https://youtu.be/0ySznjdXMEA

https://youtu.be/SxvBbXuD_sc

Sairam