CS202: IT Workshop

Java

Multithreading

Ref:

1. Herb Schildt: Java The Complete Reference, 8/e Tata Mcgraw Hill Education.

Thread

- Thread indicates a path of execution or a task
- A program (i.e. process) can have multiple threads within it; each thread does some portion of work
- Threads can run in parallel
- Java **hides** the complexities (OS level management) and provides a **rich support** for multithreading
- JVM can create multiple **thread** of execution within a program
- A thread in Java can be at different state (New, Runnable, Waiting, Terminated, Blocked)

Thread class in Java

- A thread (called *main*) runs for every Java program
- Static method ***currentThread***() of class Thread may be used to get the information

```
Thread t = Thread.currentThread();

System.out.println("Current thread: " + t);
```

- Thread class supports a number of methods

```
t.getState()

t.setName("My Thread");

Thread.activeCount()
```

Returns the state of the thread t

We can give a custom name

Returns the number of active threads in the system

Creating thread in Java

- Threads in Java can be created in two ways
  - -- Extending class Thread
  - -- Implementing interface Runnable

```
class Hi extends Thread {
        . . .
        public void run() {
            . . .
        }
}
. . .
start();
```

We need to override **run()** method of Thread

This is the entry point. Execution starts here!

run() method is called implicitly

Any object of class Hi will be a thread

Thread is to be initiated by calling **start()** method

Creating thread in Java

- Another way to create thread is to implement Runnable

```
class Hello implements Runnable {
        . . .
        Thread t = new Thread( . . . );
        t.start();
        . . .
        public void run() {
            . . .
        }
}
```

We need to create a thread passing Runnable instance and to start it

start() method will make a call to run()

Execution of the thread will begin from run()

We need to define the abstract method run() of interface Runnable

- Thread creation by making an object

```
new Hello();
```

Code (ThreadDemoRunnable.java): Screen share

Controlling thread execution

- In our programs, main thread runs along with additional child threads
- They were executing independently and finished independently!
- **We often want that main thread should end only after all child threads end. How to achieve that?**
- We can make the main thread **wait** by introducing **sleep**

```
Thread.sleep(5000);
```

Thread sleeps for 5000 millisecond

- Better way is to use join() method

```
obj1.t.join();
```

t1 is an object;
t is the name of the Thread variable inside class

Code (ThreadJoinDemo.java): Screen share

Handling multiple threads

- We can create multiple threads by creating multiple objects and starting them by calling start() method

```
ThreadRun obj1 = new ThreadRun("one");

ThreadRun obj2 = new ThreadRun("two");
```

Code (ThreadDemoMultiple.java, ThreadHiHello.java): Screen share

- When two or more methods accesses any shared item, they must ensure **consistency**!
  - Only one thread will be allowed to execute a method / set of statements
  - Java supports synchronization

```
synchronized void increment() { . . . }
```

Only one thread can execute the method at a time

Code (ThreadSyncDemo.java): Screen share

Inter thread communication

- Synchronization ensures that only one thread executes a code segment at any time
- This makes the other threads keeps on waiting → wastes CPU time
- A thread can communicate with another using system defined methods:
  - ***wait():*** forces the current thread to wait until some other thread invokes notify() or notifyAll() on the same object.
  - ***notify():*** is used for waking up a thread that are waiting for an access to this object's monitor.
  - ***notifyAll()***: wakes all threads that are waiting on this object's monitor.

Code (PCDemo.java, PCDemoNotify.java**): Screen share**

Questions?