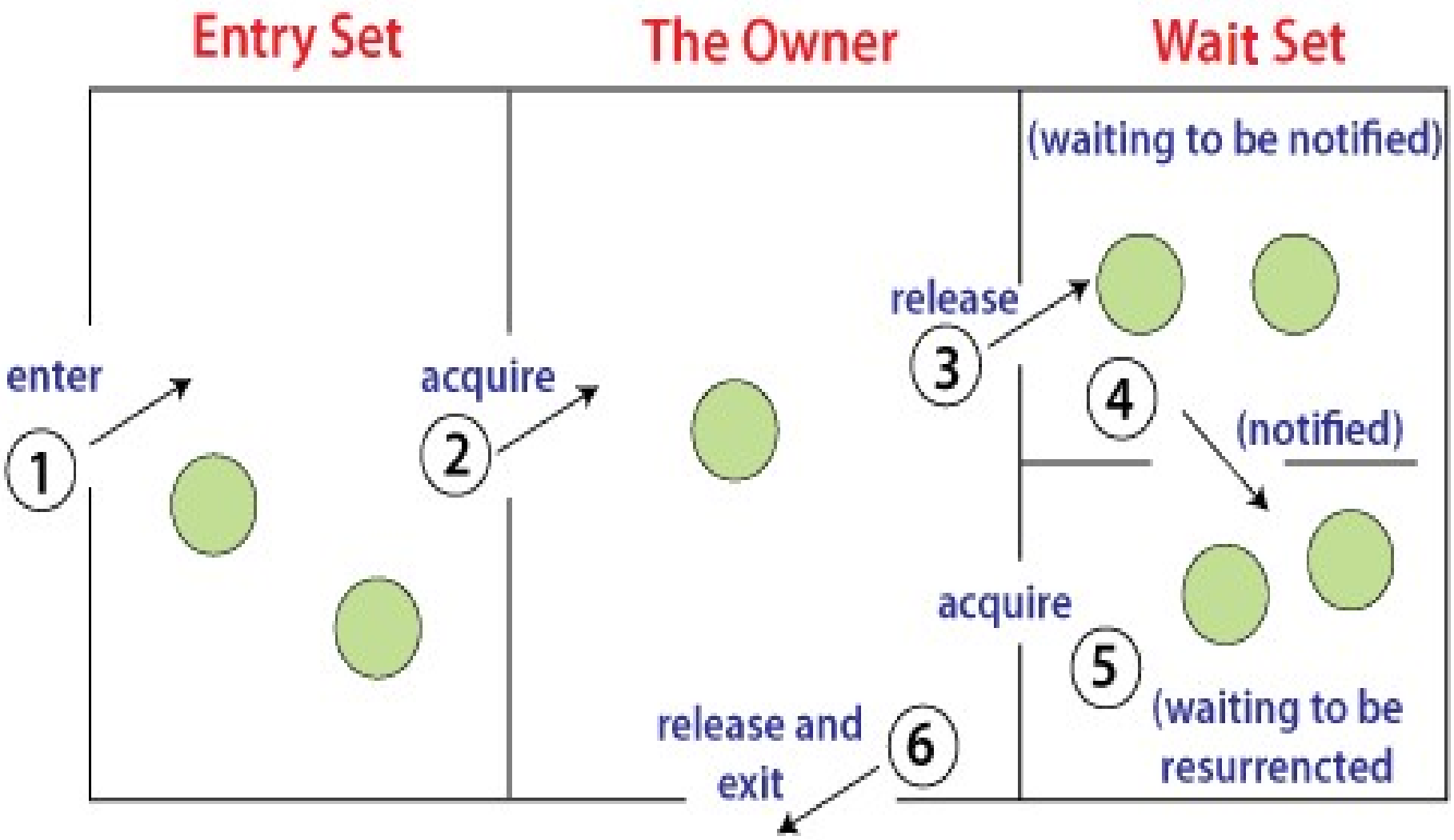
Inter-thread Communication

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University

- **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:
- wait()
- notify()
- notifyAll()

Inter-thread communication



1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

wait() method

- Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

```
public final void notify()
```

notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:

```
public final void notifyAll()
```

Wait() vs sleep()

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{
wait();
}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
```

```
synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
```

```
public void run(){
c.withdraw(15000);
} }
.start();
new Thread(){
public void run(){
c.deposit(10000);
} }
.start();
 }
}
```

Output: going to withdraw…
Less balance; waiting for deposit…
going to deposit…
deposit completed…
withdraw completed

Join()

- The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Syntax:

- public void join()throws InterruptedException
- public void join(long milliseconds)throws InterruptedException

```
class TestJoinMethod1 extends Thread{
public void run(){
for(int i=1;i<=5;i++){
try{
 Thread.sleep(500);
}catch(Exception e){
System.out.println(e);
}
System.out.println(i);
}
}
```

```
public static void main(String args[]){
TestJoinMethod1 t1=new TestJoinMethod1();
TestJoinMethod1 t2=new TestJoinMethod1();
TestJoinMethod1 t3=new TestJoinMethod1();
t1.start();
try{    t1.join();
 }catch(Exception e){
System.out.println(e);
}
t2.start();
t3.start();
}
}
```

Output:1

2

3

4

5

1

1

2

2

3

3

4

4

5

5

Example of join(long miliseconds)

```
class TestJoinMethod2 extends Thread{
public void run(){
for(int i=1;i<=5;i++){
try{
Thread.sleep(500);
}catch(Exception e){
System.out.println(e);
}
System.out.println(i);
 }
}
```

```
public static void main(String args[]){
TestJoinMethod2 t1=new TestJoinMethod2();
TestJoinMethod2 t2=new TestJoinMethod2();
TestJoinMethod2 t3=new TestJoinMethod2();
t1.start();
try{
t1.join(1500);   }
catch(Exception e){
System.out.println(e);
}
t2.start();
 t3.start();
}
}
```

Output:

1
2
3
1
4
1
2
5
2
3
3
4
4
5
5

isAlive()

- isAlive( ) method returns true if the thread upon which it is called is still running. It returns false otherwise.

```
class NewThread implements Runnable {
  String name; // name of thread
  Thread t;

  NewThread(String threadname) {
    name = threadname;
    t = new Thread(this, name);
    System.out.println("New thread: " + t);
    t.start(); // Start the thread
  }

  // This is the entry point for thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {
      System.out.println(name + " interrupted.");
    }
    System.out.println(name + " exiting.");
  }
}
```

```
class DemoJoin {
  public static void main(String args[]) {
    NewThread ob1 = new NewThread("One");
    NewThread ob2 = new NewThread("Two");
    NewThread ob3 = new NewThread("Three");

    System.out.println("Thread One is alive: "
                       + ob1.t.isAlive());
    System.out.println("Thread Two is alive: "
                       + ob2.t.isAlive());
    System.out.println("Thread Three is alive: "
                       + ob3.t.isAlive());
    // wait for threads to finish
   try {
      System.out.println("Waiting for threads to finish.");
      ob1.t.join();
       ob2.t.join();
       ob3.t.join();
     } catch (InterruptedException e) {
       System.out.println("Main thread Interrupted");
     }
```

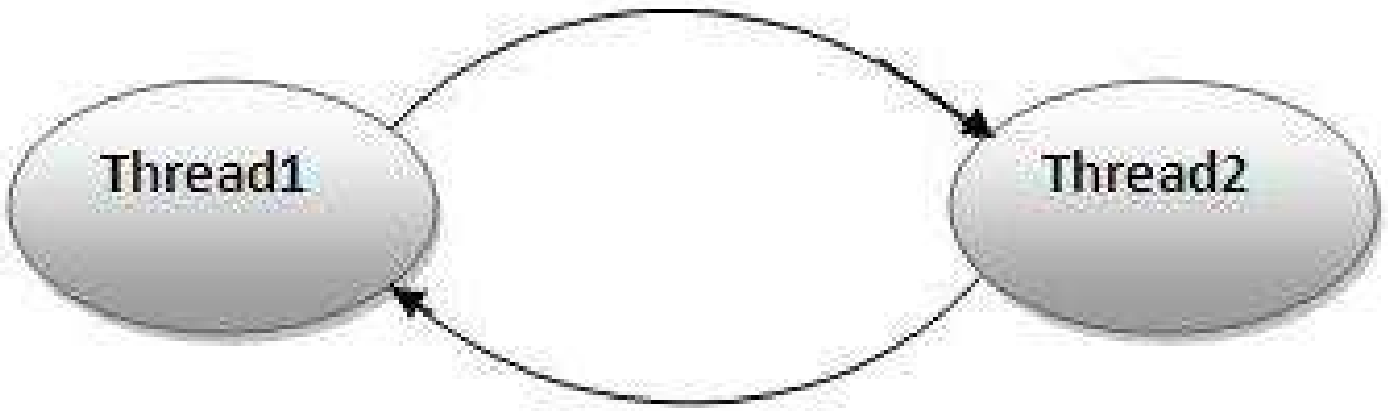
```
        System.out.println("Thread One is alive: "
                           + ob1.t.isAlive());
        System.out.println("Thread Two is alive: "
                           + ob2.t.isAlive());
        System.out.println("Thread Three is alive: "
                           + ob3.t.isAlive());

        System.out.println("Main thread exiting.");
    }
}
```

```
New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
New thread: Thread[Three,5,main]
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for threads to finish.
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
One: 3
Two: 3
Three: 3
One: 2
Two: 2
Three: 2
One: 1
Two: 1
Three: 1
```

```
Two exiting.
Three exiting.
One exiting.
Thread One is alive: false
Thread Two is alive: false
Thread Three is alive: false
Main thread exiting.
```

Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



```
public class TestDeadlockExample1 {
public static void main(String[] args) {
final String resource1 = “I am resource 1";
final String resource2 = “I am resource 2";
// t1 tries to lock resource1 then resource2
 Thread t1 = new Thread() {
public void run() {
synchronized (resource1) {
System.out.println("Thread 1: locked resource 1");
 try {Thread.sleep(100);} catch (Exception e) {}
 synchronized (resource2) {
System.out.println("Thread 1: locked resource 2");
}        }      }     };
```

```
// t2 tries to lock resource2 then resource1
Thread t2 = new Thread() {
public void run() {
synchronized (resource2) {
System.out.println("Thread 2: locked resource 2");
try {
Thread.sleep(100);
} catch (Exception e) {}
synchronized (resource1) {
System.out.println("Thread 2: locked resource 1");
}
}      }      };
```

```
t1.start();
t2.start();
} }
```

Output: Thread 1: locked resource 1

Thread 2: locked resource 2

Thank you