# 10.4.4 얕은복사(계속)

- 자바에서 객체복사에 대한 정리
  - 자바에서 **객체복사 방법은 clone() 메서드를 사용하는 방법밖에는 없다.**
  - 그리고 우리가 객체를 메모리 차원에서 완벽하게 복사하기 위해서는 **Cloneable 인터페이스를 구현해야만** 한다.
  - 이 때 clone()의 접근지정자가 protected이기 때문에 외부 접근이 불가능하다.
  - 이를 해결하기 위해서 내부에서 **super로 접근한다는 것에 주의**
  - 자바의 객체복사는 Cloneable이란 인터페이스와 Object의 clone()을 이용해서 구현할 수 있다.
  - 프로그램적으로 나타나는 얕은 복사의 현상 때문에 프로그래머가 직접 깊은 복사를 구현해야 한다.

# 10.5 wait()와 notify()

# 10.5.1 스레드의 동기화와 제어

- 동기화의 제어
  - Thread에 대해 어느 정도 알고 나면 반드시 만나는 약방의 감초가 바로 wait()와 notify()이다.
  - wait()와 notify()는 객체의 **메모리**와 직접적인 관련이 있으며, 데이터의 **동기화**와 많은 관련이 있다.

# 10.5.1 스레드의 동기화와 제어(계속)

- 동기화는 **스레드**와 직접적인 관련이 있다.
  - 스레드가 없다면 동기화 문제도 제기되지 않을 것이다.
- 공유자원
  - 동시에 **여러 개의 스레드가** 하나의 자원을 상대로 작업을 할 때 이 자원은 공유자원이 된다.
- 스레드의 동기화
  - 하나의 스레드가 공유자원을 차지했다면 다른 스레드들을 대기하게 만드는 것.
  - 차례대로 자원을 사용할 때 동기화가 보장되었다고 한다.

# 10.5.1 스레드의 동기화와 제어(계속)

- synchronized 키워드
  - 해당 공유자원에 락(Lock)를 걸기 위해서 synchronized 키워드를 사용한다.

- 공유자원을 synchronized로 동기화했을 때
  - 하나의 스레드가 공유자원을 사용할 때
    - 자동으로 공유자원에 락(Lock)을 걸어준다.
    - 공유자원을 원하는 다른 스레드들은 자동으로 대기하게 된다.
  - 공유자원의 락(Lock)이 해제되었을 때
    - 대기하던 다른 스레드들은 순서대로 공유자원을 사용하게 된다.

- synchronized
  - 자동 동기화 키워드

# 10.5.1 스레드의 동기화와 제어(계속)

- 비디오 테이프를 빌릴 경우의 대기 메커니즘
  - 비디오 가게에 비디오 테이프가 있는지 고객이 가게에 가서 확인해야 한다.
  - 비디오 테이프가 없다면 계속해서 확인해야 한다.
  - 비디오 테이프가 있다면 빌려가면 된다.

- 결론
  - 결론적으로 스레드가 지속적으로 원하는 자원이 있는지 체크하는 메커니즘이 된다.
    - 한마디로 말하면 부지런한 고객 메커니즘이 되는 것이다.

## §10-15 DiligentClientMain.java

```
01:  /**
02:  부지런한 고객 메커니즘을 적용한 세마포어
03:  **/
04:  import java.util.*;
05:  class VideoShop{
06:     private Vector buffer = new Vector();
07:     public VideoShop(){
08:        buffer.addElement("은하철도999-0");
09:        buffer.addElement("은하철도999-1");
10:        buffer.addElement("은하철도999-2");
11:     }
12:     public synchronized String lendVideo(){
13:        if(buffer.size()>0){
14:           String v = (String)this.buffer.remove(buffer.size()-1);
15:           return v;
16:        }else{
17:           return null;
18:        }
19:     }
20:     public synchronized void returnVideo(String video){
21:        this.buffer.addElement(video);
22:     }
23:  } //end of VideoShop class
```

```
25:  class Person extends Thread{
26:     public void run(){
27:        try{
28:           String v;
29:           while((v = DiligentClientMain.vShop.lendVideo()) == null ){
30:              System.out.println(this.getName() + "비디오가 없군요! 나중에 와야지!");
31:              this.sleep(200);
32:           }
33:           System.out.println(this.getName() + ":" + v  + " 대여");
34:           System.out.println(this.getName() + ":" + v  + " 보는중");
35:           this.sleep(5000);
36:           System.out.println(this.getName() + ":" + v  + " 반납");
37:           DiligentClientMain.vShop.returnVideo(v);
38:        }catch(InterruptedException e){e.printStackTrace();}
39:     }
40:  } //end of Person class
42:  public class DiligentClientMain{
43:     public static VideoShop vShop = new VideoShop();
44:     public static void main(String[] args){
45:        System.out.println("프로그램 시작");
46:        Person p1 = new Person();   Person p2 = new Person();
48:        Person p3 = new Person();   Person p4 = new Person();
50:        p1.start(); p2.start(); p3.start();  p4.start();
54:        System.out.println("프로그램 종료");
55:     } //end of main
56:  } //end of DiligentClientMain class
```

```
C:\javasrc\chap10>javac DiligentClientMain.java
C:\javasrc\chap10>java DiligentClientMain
프로그램 시작
프로그램 종료
Thread-0:은하철도999-2 대여
Thread-0:은하철도999-2 보는중
Thread-1:은하철도999-1 대여
Thread-1:은하철도999-1 보는중
Thread-2:은하철도999-0 대여
Thread-2:은하철도999-0 보는중
Thread-3비디오가 없군요! 나중에 와야지!
Thread-3비디오가 없군요! 나중에 와야지!
Thread-3비디오가 없군요! 나중에 와야지!
.........
중간 생략
.........
Thread-3비디오가 없군요! 나중에 와야지!
Thread-3비디오가 없군요! 나중에 와야지!
Thread-3비디오가 없군요! 나중에 와야지!
Thread-0:은하철도999-2 반납
Thread-1:은하철도999-1 반납
Thread-2:은하철도999-0 반납
Thread-3:은하철도999-0 대여
Thread-3:은하철도999-0 보는중
Thread-3:은하철도999-0 반납
```

# 10.5.1 스레드의 동기화와 제어(계속)

- 설명
  - 비디오 가게에는 세 개의 비디오 테이프만 존재한다.
  - 하지만 네 사람의 고객이 비디오 테이프를 원하고 있다.
  - 세 사람만 비디오 테이프를 빌릴 것이며, 나머지 한 사람은 지속적으로 비디오 가게를 방문해서 비디오 테이프가 있는지 체크하게 될 것이다.
  - 그래서 다음과 같이 Person의 run() 메서드에 반복적으로 비디오 가게에 비디오가 있는지 체크하고 있다.

- String v;
  - while((v = DiligentClientMain.vShop.lendVideo()) == null ){
  - System.out.println(this.getName() + "비디오가 없군요! 나중에 와야지!");
  - this.sleep(200); //0.2초마다 한번씩 비디오 테이프가 있는지 확인
  - }

# 10.5.1 스레드의 동기화와 제어(계속)

- 세마포어(Semaphore)
  - 세마포어(Semaphore)란 자원이 제한되어 있으며 **제한된 자원의 개수 만큼만 자원을 운영**하는 것이다.

- 비디오 가게
  - 여러분의 비디오 테이프가 있다면 while문을 빠져 나와서 비디오 테이프를 빌려가게 될 것이다.
  - 이러한 **자원의 활용** 메커니즘을 세마포어(Semaphore)라고 한다.

# 10.5.2 wait()와 notify()

- 세마포어는 두 가지 관점
  - 첫 번째
    - **스레드** 측에서 자원이 있는지 없는지 지속적으로 체크하는 것이다.
    - 이렇게 되면 약간의 **시스템적인 부하**를 초래할 수 있다.
  - 두 번째
    - 비디오 가게에 비디오 테이프가 없을 경우 **주인**이 다음과 같이 말한다고 생각해 보죠.
      - 지금은 비디오 테이프가 없다. 집에 가서 대기(**wait**)하시오.
      - 비디오 테이프가 들어오면 전화로 알려(**notify**) 드리겠다.

# 10.5.2 wait()와 notify()(계속)

- 친절한 비디오 가게 아저씨
  - 비디오 테이프가 들어왔을 때 전화로 알려 준다면 전체적인 비용은 절감될 것이다.
  - 이러한 메커니즘은 앞에서 배운 부지런한 고객 메커니즘과 상반되는 경우이다.
  - 필자는 이 메커니즘을 친절한 비디오 가게 아저씨 메커니즘이라고 말하고 있다.

- 세마포어의 메커니즘
  - 부지런한 고객 메커니즘
  - 친절한 비디오 가게 아저씨 메커니즘

# 10.5.2 wait()와 notify()(계속)

- 친절한 비디오 가게 아저씨를 구현하기 위한 메소드

  - wait()
    - 스레드를 NotRunnable 상태로 만든다.

  - notify(), notifyAll()
    - NotRunnable 상태의 스레드를 Runnable 상태로 만든다.

# 10.5.2 wait()와 notify()(계속)

- NotRunnable 상태로 만드는 두 가지 방법
  - 첫 번째
    - sleep()은 주어진 시간만큼만 NotRunnable 상태가 되며, 시간이 지나면 자동으로 Runnable 상태가 된다.
  - 두 번째
    - wait()이다.
    - wait()로 NotRunnable 상태를 만들었다면 notify()나 notifyAll()을 해주기 전까지는 Runnable 상태로 되돌아 올 수 없다.
    - notify()는 Not Runnable 상태의 스레드를 Runnable 상태로 만드는 역할을 한다.

- wait()와 notify() 메서드의 예제
  - 다음의 예제는 앞에서 나온 부지런한 고객 메커니즘을 수정해서 친절한 비디오 가게 아저씨 메커니즘으로 변경한 것이다.

**§10-16 GoodVideoStoreKeeperMain.java**

```
01:  /**
02:  친절한 비디오 가게 아저씨 메커니즘을 적용한 세마포어의 구현
03:  **/
04:  import java.util.*;
05:  class VideoShop{
06:     private Vector buffer = new Vector();
07:     public VideoShop(){
08:        buffer.addElement("은하철도999-0");
09:        buffer.addElement("은하철도999-1");
10:     }
11:     public synchronized String lendVideo() throws InterruptedException{
12:        Thread t = Thread.currentThread();
13:        if(buffer.size()==0){
14:           System.out.println(t.getName() + ": 대기 상태 진입");
15:           this.wait();
16:           System.out.println(t.getName() + ": 대기 상태 해제");
17:        }
18:        String v = (String)this.buffer.remove(buffer.size()-1);
19:        return v;
20:     }
21:     public synchronized void returnVideo(String video){
22:        this.buffer.addElement(video);
23:        this.notify();
24:     }
25:  } //end of VideoShop class
```

```
27:  class Person extends Thread{
28:     public void run(){
29:        try{
30:           String v = GoodVideoStoreKeeperMain.vShop.lendVideo();
31:           System.out.println(this.getName() + ":" + v  + " 대여");
32:           System.out.println(this.getName() + ":" + v  + " 보는중\n");
33:           this.sleep(1000);
34:           System.out.println(this.getName() + ":" + v  + " 반납");
35:           GoodVideoStoreKeeperMain.vShop.returnVideo(v);
36:        }catch(InterruptedException e){e.printStackTrace();}
37:     }
38:  } //end of Person class
40:  public class GoodVideoStoreKeeperMain{
41:     public static VideoShop vShop = new VideoShop();
42:     public static void main(String[] args){
43:        System.out.println("프로그램 시작");
44:        Person p1 = new Person();  Person p2 = new Person();
46:        Person p3 = new Person();  Person p4 = new Person();
48:        Person p5 = new Person();   Person p6 = new Person();
50:        p1.start();      p2.start();
52:        p3.start();      p4.start();
54:        p5.start();      p6.start();
56:        System.out.println("프로그램 종료");
57:     } //end of main
58:  } //end of GoodVideoStoreKeeperMain class
```

```
C:\javasrc\chap10>javac GoodVideoStoreKeeperMain.java
C:\javasrc\chap10>java GoodVideoStoreKeeperMain
프로그램 시작
프로그램 종료
Thread-0:은하철도999-1 대여
Thread-0:은하철도999-1 보는중

Thread-1:은하철도999-0 대여
Thread-1:은하철도999-0 보는중

Thread-2: 대기 상태 진입
Thread-3: 대기 상태 진입
Thread-4: 대기 상태 진입
Thread-5: 대기 상태 진입
Thread-0:은하철도999-1 반납
Thread-2: 대기 상태 해제
Thread-2:은하철도999-1 대여
Thread-2:은하철도999-1 보는중

Thread-1:은하철도999-0 반납
Thread-3: 대기 상태 해제
Thread-3:은하철도999-0 대여
Thread-3:은하철도999-0 보는중

Thread-2:은하철도999-1 반납
Thread-4: 대기 상태 해제
Thread-4:은하철도999-1 대여
Thread-4:은하철도999-1 보는중

Thread-3:은하철도999-0 반납
Thread-5: 대기 상태 해제
Thread-5:은하철도999-0 대여
Thread-5:은하철도999-0 보는중

Thread-4:은하철도999-1 반납
Thread-5:은하철도999-0 반납
```

# 10.5.2 wait()와 notify()(계속)

- static 변수 선언
  - 비디오 가게를 구현한 VideoShop 객체를 프로그램에서 단 하나만 생성하기 위해서 다음과 같이 선언하고 있다.
    - public static VideoShop vShop = new VideoShop();
- static의 사용
  - 비디오 가게를 사용하기 위해서는 외부에서 다음과 같이 접근할 수 있을 것이다.(클래스 이름으로)
    - GoodVideoStoreKeeperMain.vShop

# 10.5.2 wait()와 notify()(계속)

- 설명
  - vShop이 public으로 선언되어 있으며 static으로 선언되어 있기 때문에 클래스의 이름으로 vShop에 접근할 수 있다.
  - main에서는 6개의 Person 객체를 만든 후 스레드의 형식으로 start() 시키고 있다.
  - Person에서 하는 작업
    - Person의 작업은 vShop에서 비디오를 빌려서 감상한 후 반환하면 된다.
  - 하지만 6명의 Person이 동시에 시작하게 된다.

# 10.5.2 wait()와 notify()(계속)

- 설명
  - 6명의 Person이 동시에 비디오 테이프를 빌리려고 한다.
  - 하지만 vShop에 비디오는 2개밖에 없다.
  - 생성자에서 2개의 비디오 테이프를 보유한 상태이다.

- VideoShop의 생성자

```
class VideoShop{
private Vector buffer = new Vector();
        public VideoShop(){
            buffer.addElement("은하철도999- 0");
            buffer.addElement("은하철도999- 1");
        }
        //.....
}
```

# 10.5.2 wait()와 notify()(계속)

- wait()
  - 2개의 비디오 테이프가 존재하지만 lendVideo()를 호출하는 Person은 6명이다.
  - 그렇다면 2명은 우선적으로 빌려 줄 수 있을 것이다.
  - 하지만 나머지 4명은 비디오 테이프가 반환될 때까지 wait()를 시키게 된다.

- public synchronized String lendVideo() throws InterruptedException{

```
        if(buffer.size()==0){
            this.wait();
        }
        //비디오 대여
}
```

# 10.5.2 wait()와 notify()(계속)

- notify()
  - 만약 returnVideo(String video)를 통해서 비디오 테이프가 반납이 된다면 notify()를 통해서 알려주는 것이다.

- public synchronized void returnVideo(String video){
  - //비디오 테이프 반납
  - this.notify();
  - }

# 10.5.2 wait()와 notify()(계속)

- 설명
  - notify()의 호출로 wait()되었던 스레드 중에 우선 순위가 높은 스레드가 깨어나게 될 것이다.

- synchronized 와 wait(), notify(), notifyAll()
  - 자바에서 공유 자원의 동기화 문제를 해결하기
    - synchronized
      - 미세한 부분의 작업은 제어가 힘들다.
    - wait(), notify(), notifyAll()
      - 미세한 동기화를 제어할 때 사용

# 10.5.2 wait()와 notify()(계속)

- wait()
  - wait() 메서드는 스레드가 동기화된 공유자원을 요청할 때 스레드를 **대기상태**로 만들 수 있다.
- notify()
  - notify() 메서드는 대기 중에 있는 스레드에게 동기화된 공유자원을 사용할 수 있도록 스레드를 **실행 상태**로 만들어 주는 역할을 한다.
- wait(), notify()
  - wait(), notify() 메서드를 이용한다면 스레드를 보다 효율적으로 관리할 수 있다.

# 10.5.3 notifyAll()

- 기존의 notify()
  - 앞의 예제에서 비디오 테이프가 반환되었을 때 공유자원을 대기하는 스레드가 있다면 알려주기 위해서 notify()를 사용했다.

```
public synchronized void returnVideo(String video){
        //비디오 테이프 반납
        this.notify();
}
```

- 문제점
  - 위가 같이 했을 때 notify()를 하면 대기 중에 있는 **하나**의 스레드를 골라서 깨워주게 된다.
    - buffer를 사용하려고 하는 스레드 중 하나를 골라서 깨워주는 것이다.
  - 어떤 스레드를 선택해서 깨워줄지 알 수 없다고 한다.
  - notify()가 내부적으로 완벽하게 하나의 스레드를 깨워준다는 것을 **보장**할 수 **없다**고 한다.

# 10.5.3 notifyAll() (계속)

- 해결책
  - 만약 notify()가 스레드를 깨워주지 않으면 심각한 문제가 발생할 수 있다.
  - 이러한 문제를 해결하기 위해서 대기중인 스레드를 몽땅 다 깨워버리는 기법을 사용한다.
    - 즉 위의 notify() 대신에 **notifyAll()**을 사용하는 것이다.

- 비디오 테이프가 반납되었을 때 대기하는 **모든** 스레드를 깨우는 예

```
public synchronized void returnVideo(String video){
        //비디오 테이프 반납
        this.notifyAll();
}
```

# 10.5.3 notifyAll() (계속)

- 설명
  - notifyAll()을 사용하면 대기중인 모든 스레드가 깨어난다.
  - 자원이 없으면 다시 wait()시켜 버리는 메커니즘으로 변경해 주면 된다.

- 다음의 구문으로 notifyAll()에 대처할 수 없다.

```
public synchronized String lendVideo() throws InterruptedException{
        if(buffer.size()==0){
            this.wait();
        }
        //비디오 대여
}
```

- notifyAll()에 대처할 수 있는 wait()

```
public synchronized String lendVideo() throws InterruptedException{
        while(buffer.size()==0){
            this.wait();
        }
        String v = (String)this.buffer.remove(buffer.size()- 1);
        return v;
}
```

# 10.5.3 notifyAll() (계속)

- 설명
  - notifyAll ()을 이용한 메커니즘은 흡사 양치기 소년과 같은 메커니즘처럼 동작한다.
    - 모든 대기중인 스레드를 몽땅 깨워놓고 자원이 있으면 작업을 시키고 그렇지 않으면 다시 wait() 시켜 버리는 동작을 반복하게 되는 것이다.
  - notifyAll ()를 사용하면 스레드들을 상대로 작업을 할 때, 깨워주지 않을 위험에 빠질 염려는 없다.
  - 실전에서는 notify()보다 notifyAll() 방식의 프로그램을 더 많이 사용한다.

**§10-17 SheepRaisingBoyMain.java**

```
01:  /**
02:  양치기 소년 메커니즘을 적용한 세마포어의 구현
03:  **/
04:  import java.util.*;
05:  class VideoShop{
06:     private Vector buffer = new Vector();
07:     public VideoShop(){
08:        buffer.addElement("은하철도999-0");
09:        buffer.addElement("은하철도999-1");
10:     }
11:     public synchronized String lendVideo() throws InterruptedException{
12:        Thread t = Thread.currentThread();
13:        while(buffer.size()==0){
14:           System.out.println(t.getName() + ": 대기 상태 진입");
15:           this.wait();
16:           System.out.println(t.getName() + ": 대기 상태 해제");
17:        }
18:        String v = (String)this.buffer.remove(buffer.size()-1);
19:        return v;
20:     }
21:     public synchronized void returnVideo(String video){
22:        this.buffer.addElement(video);
23:        this.notifyAll();
24:     }
25:  } //end of VideoShop class
```

```
27:  class Person extends Thread{
28:     public void run(){
29:        try{
30:           String v = SheepRaisingBoyMain.vShop.lendVideo();
31:           System.out.println(this.getName() + ":" + v  + " 대여");
32:           System.out.println(this.getName() + ":" + v  + " 보는중\n");
33:           this.sleep(1000);
34:           System.out.println(this.getName() + ":" + v  + " 반납");
35:           SheepRaisingBoyMain.vShop.returnVideo(v);
36:        }catch(InterruptedException e){e.printStackTrace();}
37:     }
38:  } //end of Person class
39:
40:  public class SheepRaisingBoyMain{
41:     public static VideoShop vShop = new VideoShop();
42:     public static void main(String[] args){
43:        System.out.println("프로그램 시작");
44:        Person p1 = new Person();  Person p2 = new Person();
46:        Person p3 = new Person();  Person p4 = new Person();
48:        Person p5 = new Person();  Person p6 = new Person();
50:        p1.start();  p2.start();
52:        p3.start();  p4.start();
54:        p5.start();  p6.start();
56:        System.out.println("프로그램 종료");
57:     } //end of main
58:  } //end of SheepRaisingBoyMain class
```

```
C:\javasrc\chap10>javac SheepRaisingBoyMain.java
C:\javasrc\chap10>java SheepRaisingBoyMain
프로그램 시작
프로그램 종료
Thread-0:은하철도999-1 대여
Thread-0:은하철도999-1 보는중

Thread-1:은하철도999-0 대여
Thread-1:은하철도999-0 보는중

Thread-2: 대기 상태 진입
Thread-3: 대기 상태 진입
Thread-4: 대기 상태 진입
Thread-5: 대기 상태 진입
Thread-0:은하철도999-1 반납
Thread-2: 대기 상태 해제
Thread-2:은하철도999-1 대여
Thread-2:은하철도999-1 보는중

Thread-3: 대기 상태 해제
Thread-3: 대기 상태 진입
Thread-4: 대기 상태 해제
Thread-4: 대기 상태 진입
Thread-5: 대기 상태 해제
Thread-5: 대기 상태 진입
Thread-1:은하철도999-0 반납
Thread-3: 대기 상태 해제
Thread-3:은하철도999-0 대여
Thread-3:은하철도999-0 보는중

Thread-4: 대기 상태 해제
Thread-4: 대기 상태 진입
Thread-5: 대기 상태 해제
Thread-5: 대기 상태 진입
Thread-2:은하철도999-1 반납
Thread-5: 대기 상태 해제
Thread-5:은하철도999-1 대여
Thread-5:은하철도999-1 보는중

Thread-4: 대기 상태 해제
Thread-4: 대기 상태 진입
Thread-3:은하철도999-0 반납
Thread-4: 대기 상태 해제
Thread-4:은하철도999-0 대여
Thread-4:은하철도999-0 보는중

Thread-5:은하철도999-1 반납
Thread-4:은하철도999-0 반납
```