제9장 자바 쓰레드

9.1 Thread 기초 (1/5)

- 프로그램
  - 명령어들의 연속(a sequence of instruction)
- 프로세스 / Thread
  - 실행중인 프로그램(program in execution)

- 프로세스 생성과 실행을 위한 함수들

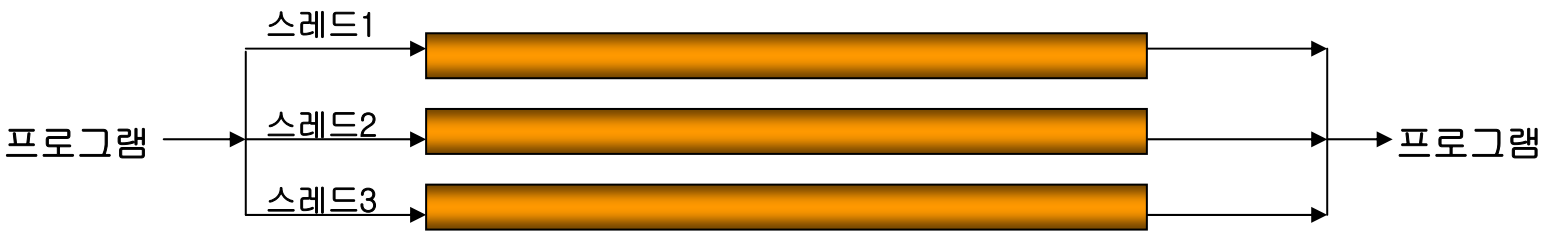
플랫폼	Win32	유닉스/리눅스	자바
프로세스 생성과 실행	CreateProcess()	fork(), exec()	Runtime.exec()

9.1 Thread 기초 (2/5)

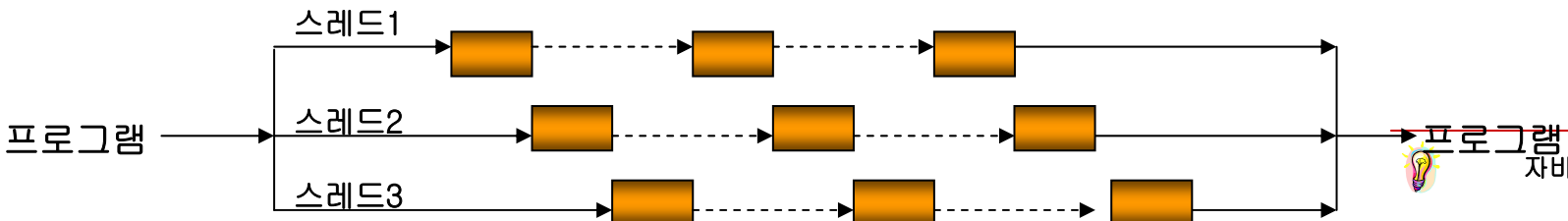
- 프로세스 단위 작업의 문제점
  - 프로세스 생성시 오버헤드
  - 컨텍스트 스위치 오버헤드
  - 프로세스간의 통신 오버헤드
  - 해결책 => 쓰레드
- 쓰레드(thread)
  - 실행될 명령어들의 연속(a sequence of instructions to be executed)
- 쓰레드의 장점
  - 생성시 적은 오버헤드
  - 주소 공간을 공유하기 때문에 자원 공유가 쉽고, 쓰레드간 통신이 쉬움
  - 컨텍스트 스위치 시 오버헤드가 적음

9.1 Thread 기초 (3/5)

- Thread
  - 순차적으로 동작하는 문장들의 단일 집합
  - 경량(lightweight) 프로세스
  - 다중 처리 기능
    - 하나의 프로그램에서 하나 이상의 객체를 생성하여 실행할 때
  - 자바는 스레드를 지원하기 위해 java.lang.Thread 클래스 제공
    - 다수개의 CPU를 가진 컴퓨터에서 다중 스레드의 실행

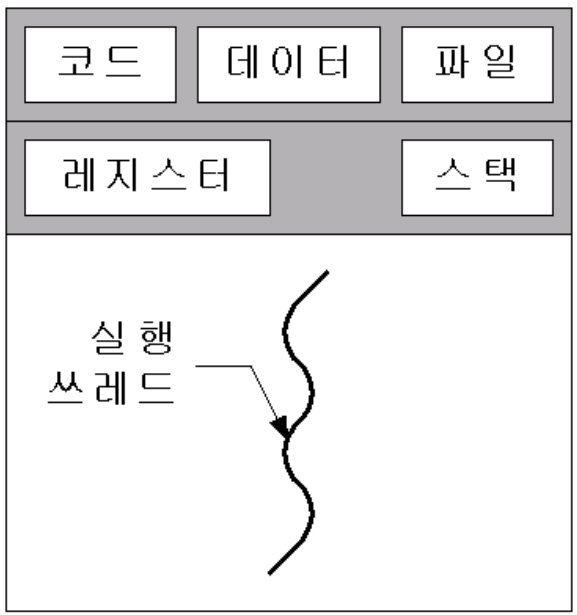


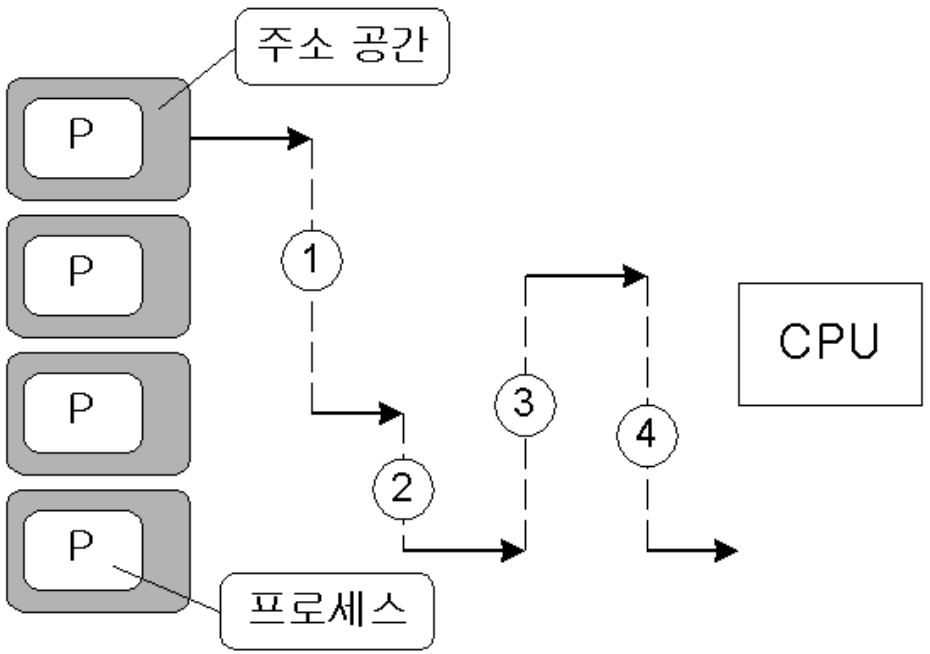
    - 한 개의 CPU를 가진 컴퓨터에서 다중 스레드의 실행



9.1 Thread 기초 (4/5)

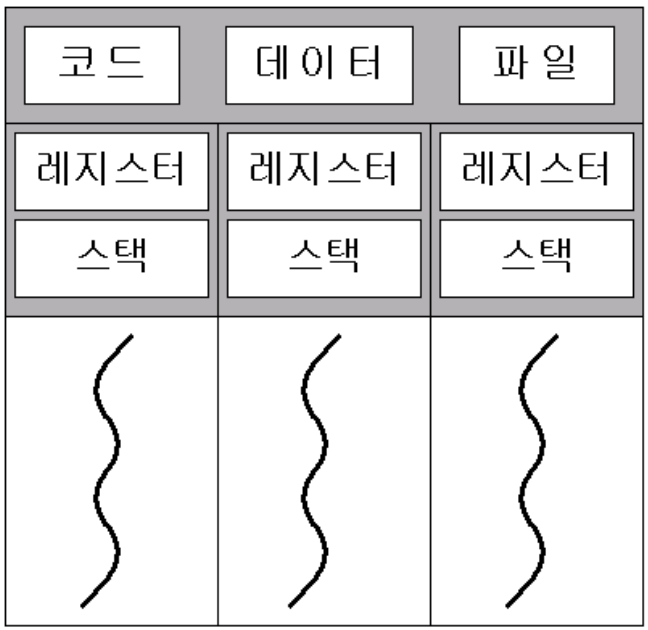
- 단일 Thread
  - 전통적인 프로세스
  - 스케쥴링 단위 = 프로세스 → 하나의 Class Instance

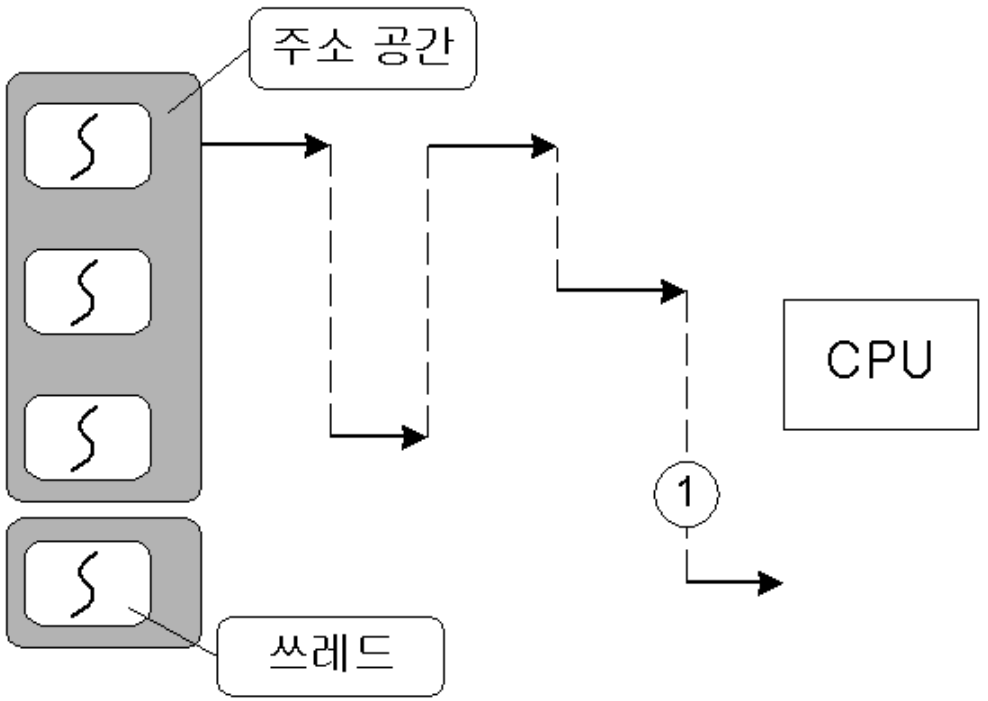




9.1 Thread 기초 (5/5)

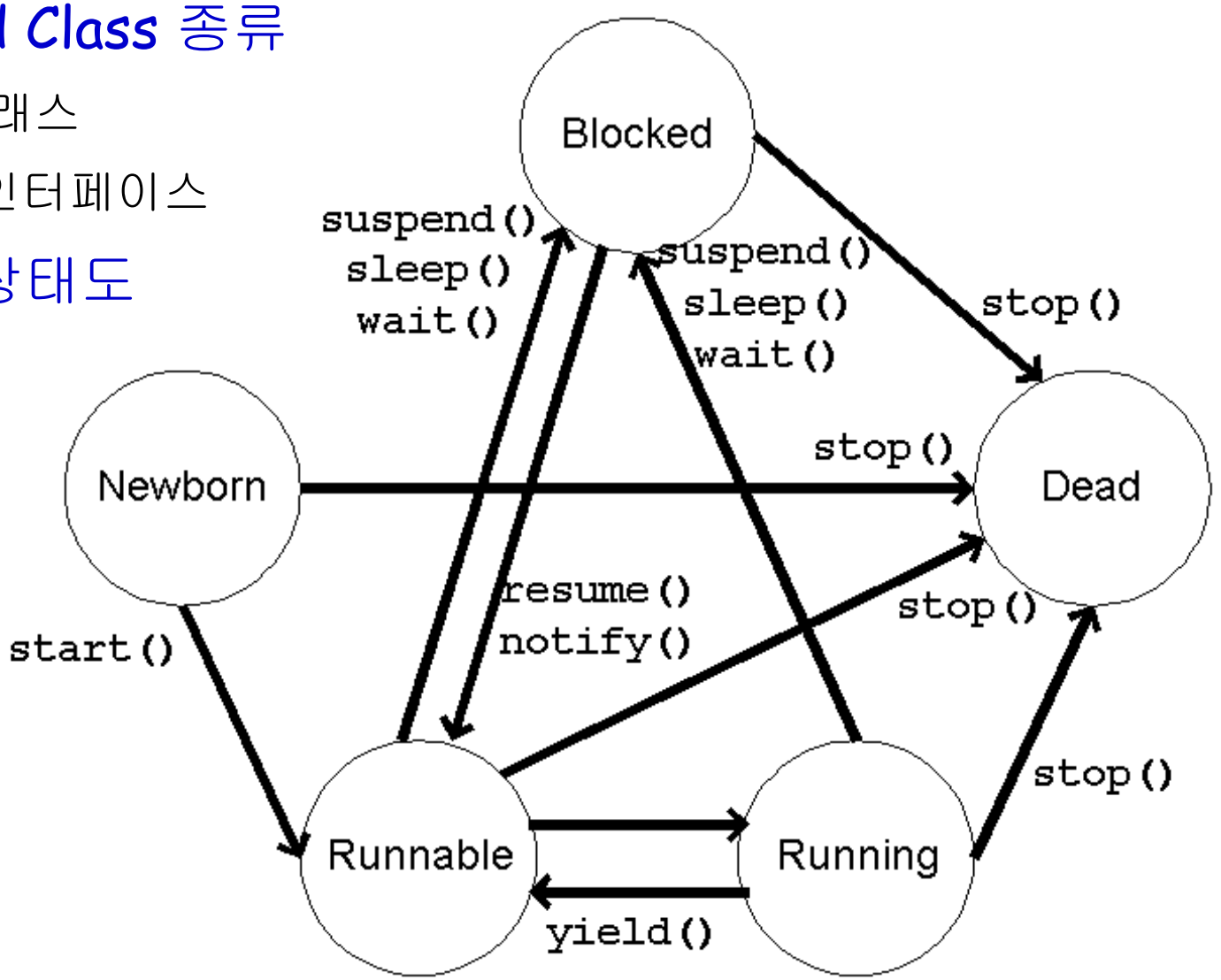
- 다중 Thread
  - 하나의 Class Instance에 여러 개의 thread가 존재
  - 스케쥴링 단위 = thread (Thread Class Instance)





9.2 자바 Thread Class (1/3)

- 자바 Thread Class 종류
  - Thread 클래스
  - Runnable 인터페이스
- 자바 Class 상태도



9.2 자바 Thread Class (2/3)

- JDK는 Thread를 지원하는 java.lang.Thread 클래스 제공
- Thread를 생성하기 위해 사용
- 4개의 생성자를 제공

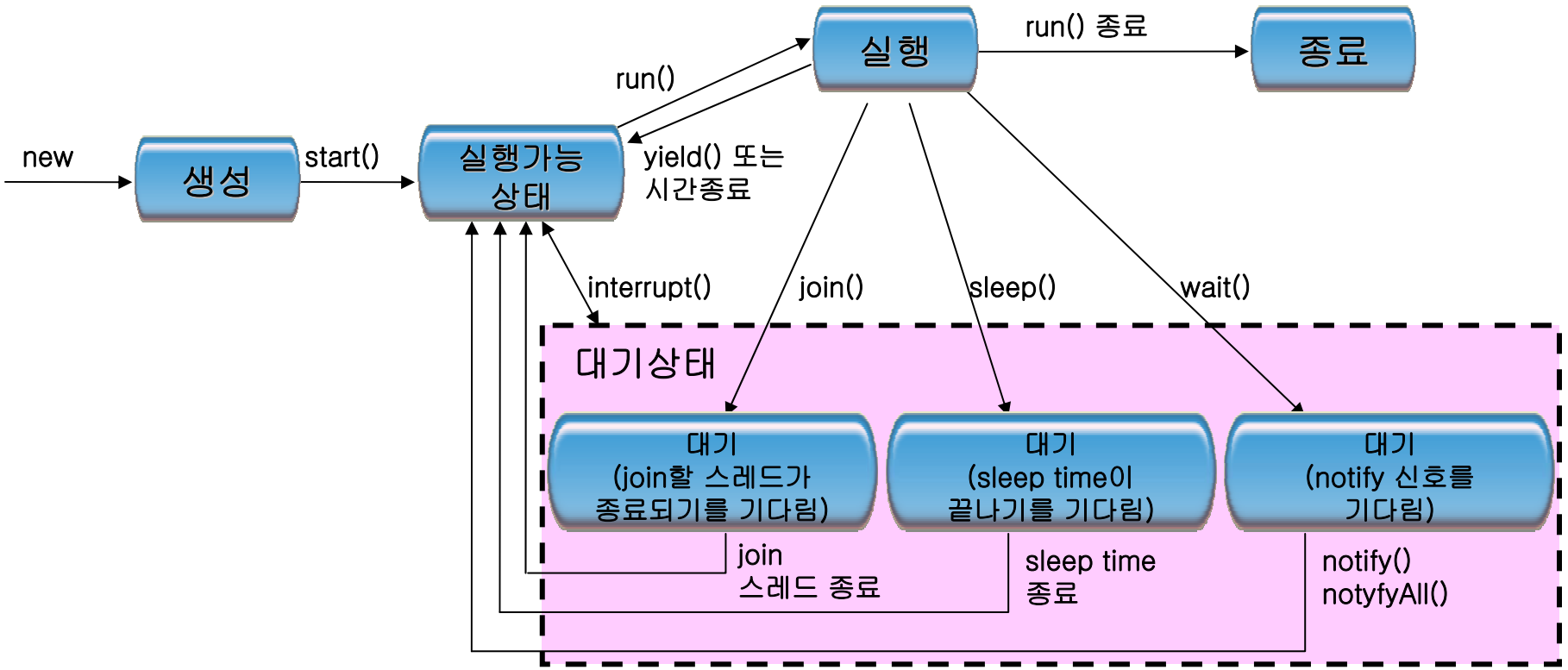
```
Thread()
Thread(String s)
Thread(Runnable r)
Thread(Runnable r, String s)
```

9.2 Java Thread 클래스의 주요 method (3/3)

메소드 이름	설명
static void sleep(long msec) throws InterruptedException	msec에 지정된 밀리초(milliseconds) 동안 대기
String getName()	스레드의 이름을 반환
void setName(String s)	스레드의 이름을 s로 설정
void start()	스레드를 시작시킨다. run() 메소드를 호출
int getPriority()	스레드의 우선 순위를 반환
void setPriority(int p)	스레드의 우선 순위를 p값으로 설정
boolean isAlive()	스레드가 시작되었고 아직 끝나지 않았으면 true를 그렇지 않으면 false를 반환
void join() throws InterruptedException	스레드가 끝날 때까지 대기
void run()	스레드가 실행할 부분을 기술하는 메소드 하위 클래스에서 오버라이딩 되어야 한다
void suspend()	스레드가 일시 정지된다. resume()에 의해 다시 시작될 수 있다

9.3 Thread 생명주기

- Java Thread instance의 생명주기



9.4 Thread 생성과 사용 (1/4)

- Thread를 생성하는 방법

방법 1) Thread 클래스로부터 직접 상속받아 Thread를 생성

→ Thread_Class extends Thread { }

→ new Thread_Class; 수행할 thread object의 생성

→ start(); : thread의 시작

→ run() { … } : thread의 수행 내용 정의

방법 2) Runnable 인터페이스를 사용하는 방법

→ 주로 현재의 클래스가 이미 다른 클래스로부터 상속 받고 있는 경우

→ 다중 상속이 되는 경우는 Interfacce Class를 Implement

→ Thread_Class extends Other_class implements Runnable { }

→ new Thread_Class; 수행할 thread object의 생성

→ start(); : thread의 시작

→ run() { … } : thread의 수행 내용 정의

9.4 Thread 생성과 사용 (2/4)

9.4.1 Thread 클래스 이용

- Thread 클래스로부터 직접 상속 받아 Thread를 생성
- Thread 클래스에서 제공되는 run() 메소드를 오버라이딩하여 생성한 Thread를 실행 시킴

```
class MyThread extends Thread  {  // Thread 클래스로부터 상속
  .........
MyThread me = new MyThread(); //Thread object를 생성
me.start();                                  // thread를 시작 시킴
.........
public void run() {                          // thread가 실행되는 내용
  .... // 상위 클래스인 Thread 클래스의 run() 메소드를 오버
  .... //라이딩하여 스레드가 수행하여야 하는 문장들을 기술
 }
  ....
}
```

9.4 Thread 생성과 사용 (3/4)

9.4.2 Runnable 인터페이스 이용

- 현재의 클래스가 이미 다른 클래스로부터 상속을 받고 있다면 Runnable 인터페이스를 사용하여 thread를 생성할 수 있다
- Runnable 인터페이스는 JDK에 의해 제공되는 라이브러리 인터페이스이다
- Runnable 인터페이스에는 run() 메소드만 정의되어 있다

```
public interface Runnable {
        public void run();
 }
```

9.4 Thread 생성과 사용 (4/4)

- 예를들어, **Applet** 클래스로부터 상속을 받고 있으므로 인터페이스 이용

```
class RunnableThead extends Applet implements Runnable {
  ……
      RunnableThread ro = new RunnableThreas(); // 객체 생성
      Thread rt = new Thread(ro);  // ro를 매개변수로 하여 thread 객체 rt를 생성
       rt.start();  // 스레드 시작

   public void run() {
     ……..  // Runnable 인터페이스에 정의된 run() 메소드를
     ……..  // 오버라이딩하여 스레드가 수행할 문장들을 기술한다
   }
 …..
}
```

9.5 Thread 생성 예제 (1/2)

- Thread 클래스로부터 상속받는Thread
  - run() 메소드를 재정의

- 예제:
  - MyTeam.java
    - 동시에 수행할 Thream의 생성 및 실행
  - Worker.java
    - Thread에 필요한 run() 함수 정의

9.5 Thread 생성 예제 (2/2)

- Runnable 인터페이스 구현

- 예제: TCarFactory.java
  - TCarFactory Class
    - 자체적으로 수행할 기능을 정의하고 있음.
      - Operate() 함수 부분
    - run() : Thread로 실행 부분을 재 작성

9.6 Thread 종료

- Thread 종료
  - 쓰레드는 **run()** 메소드가 끝나면, 종료된다
  - 무한 반복에서 쓰레드를 종료하는 방법
    - 반복문의 조건을 이용

```
while(stop == false) {
    ...
}
```

    - **break** 이용

```
while(true) {
    ...
    if(test)
        break;
}
```

9.7 Thread의 동시성

- 쓰레드의 일시 중단
  - suspend() - 쓰레드를 잠시 중단시킴
  - resume() - 중단된 쓰레드를 다시 시작시킴
- 예제: SuspendTest.java

```
 7          ma = new Thread(this, "마당쇠");
 8          ma.start();
 9          try {
10              Thread.sleep(5000);
11              System.out.println("쓰레드를 중지시킵니다.");
12              ma.suspend();
13              Thread.sleep(3000);
14              System.out.println("쓰레드를 다시 시작시킵니다.");
15              ma.resume();
```

9.8 Thread Scheduling

- Thread 스케쥴링
  - 우선 순위 선점(preemptive) 방식
  - 우선 순위 값 : 0 ~ 10
- 예제: PriorityTest.java

```
13          Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
14          ma.start();
...
18              System.out.println("마당쇠 우선순위를 +2 증가");
19              ma.setPriority(ma.getPriority() + 2);
20              Thread.sleep(3000);
21              System.out.println("돌쇠 우선순위를 +4 증가");
22              dol.setPriority(dol.getPriority() + 4);
23              Thread.sleep(5000);
```

9.9 Thread 동기화

- 동기화
  - synchronized 키워드 사용
- 예제: LockTest.java

```
23      public synchronized void doCriticalJob() {
24          String name = Thread.currentThread().getName();
25          System.out.print("[" + name + "₩t");
26          try {
27              int time = (int)(Math.random() * 2000);
28              Thread.sleep(time);
29          } catch(Exception e) { }
30          System.out.println("₩t" + name + "]");
31      }
```

9,10 Thread 응용 예

- 생산자와 소비자