과목명	객체지향프로그래밍	분반	x	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
과제명: 멀티쓰레드					

1, 과제설명 (사용자 요구사항 기술: 과제에 대한 설명 및 목표)

과제 내용

Factorial의 합을 병렬로 계산하는 fact 프로그램을 작성하시오.

- fact는 argument로 숫자 하나를 받고, 그 숫자까지의 factorial의 합을 구한다.
  - 예) ./fact 4
  - 1! + 2! + 3! + 4! 의 총 합을 구한다.
- pthread를 이용하여 각 thread가 각각 1!, 2!, 3!, 4!를 계산하고 출력한다.
- 계산이 모두 종료된 이후, 각 계산결과를 합산하고 출력한다.
- 각 thread가 공유된 변수에 접근할 때는 반드시 mutex를 이용하여 critical section을 보호하여야 한다.

주의사항 1
출력되는 순서는 상관없음. 단, total은 마지막에 출력
total값이 항상 일정하도록 thread 동기화 필요

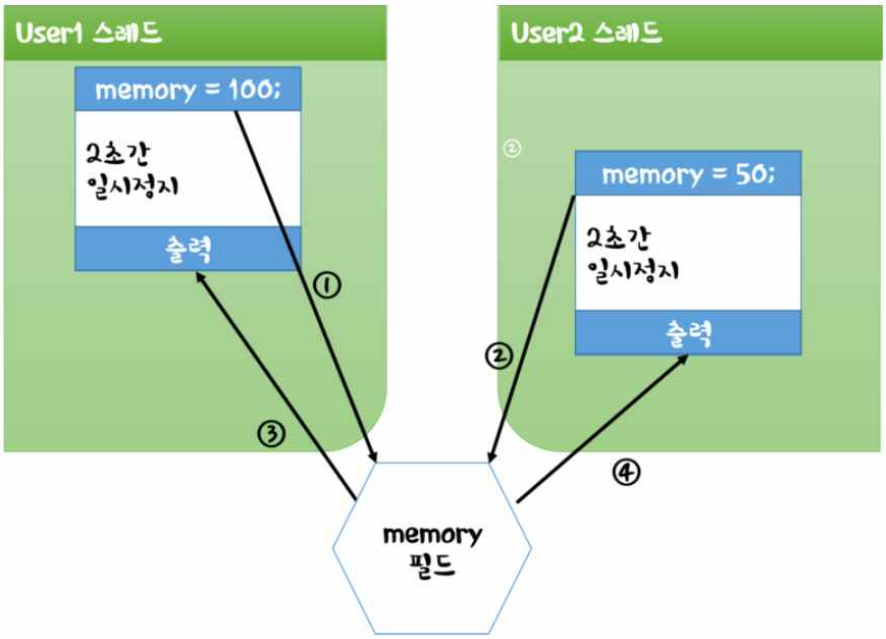
2, 사용자 요구사항을 정형적 방법으로 기술 (UML, Pseudo code, 그림등을 이용하여 기술)

static의 활용

1. 전역 변수와 전역 함수를 만들 때 활용

- 캡슐화 원칙
  - 자바에서는 C/C++와 달리 어떤 변수나 함수도 클래스 바깥에 존재할 수 없으며 클래스의 멤버로 존재하여야 한다.
- 그러나 응용프로그램 작성 시 모든 클래스에서 공유하는 전역 변수(global variable)나 모든 클래스에서 언제든지 호출할 수 있는 전역 함수(global function)를 만들어 사용할 필요가 생긴다.
  - static은 이런 문제의 해결책이다.
  - Ex) java.lang.Math 클래스
    - JDK와 함께 배포되는 클래스
    - 객체를 생성하지 않고 바로 호출할 수 있는 수학 계산용 상수와 메서드를 제공

```
public class Math {
public static int abs(int a);
public static double cos(double a);
... // 모든 멤버가 static
}
```




```
public synchronized void method() {
//임계 영역 단 하나의 스레드만 실행
}

synchronized(공유객체) {
임계영역
}
```

3. 알고리즘 및 자료구조 설계 내용

argument로 숫자 하나를 입력 받고, 입력 받은 수 만큼 객체를 생성하도록 하였다. 객체 배열의 레퍼런스 변수를 만들고, for문을 통해 객체를 생성하였다. 각 수의 factorial을 구하기 위해 각 수의 값을 store() 메소드를 통해 저장하고 start() 메소드를 통해 스레드를 시작하게 하였다. JVM에 의해 run() 메소드가 호출되어 실행되었다. run() 메소드를 오버라이딩하여 factorial을 구하는 메소드를 포함한 클래스의 객체를 생성하고, factorial을 구하는 메소드를 실행하여 각 값의 factorial 값을 출력시켰다. 그리고 그 값을 전역변수 sum에 더하고 저장했다. join() 메소드를 통해 다른 스레드의 종료를 기다린 후 sum 값을 출력하였다.

4. 소스코드 설명 (직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명)

```
MultiThread[] mt = new MultiThread[key];
for(int i=0;i<key;i++) {
mt[i]=new MultiThread();
mt[i].store(i+1);
mt[i].start();
}
```

객체 배열의 레퍼런스 변수를 생성하고, for문을 통해 객체를 생성한다. 그리고 start() 메소드를 호출하여 스레드를 시작하게 한다.

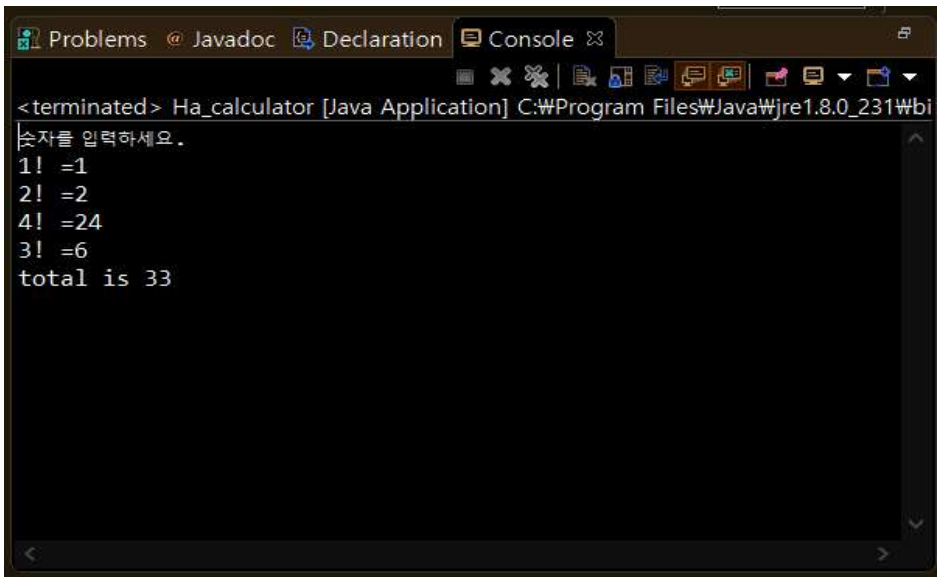
```
static int sum=0;
```

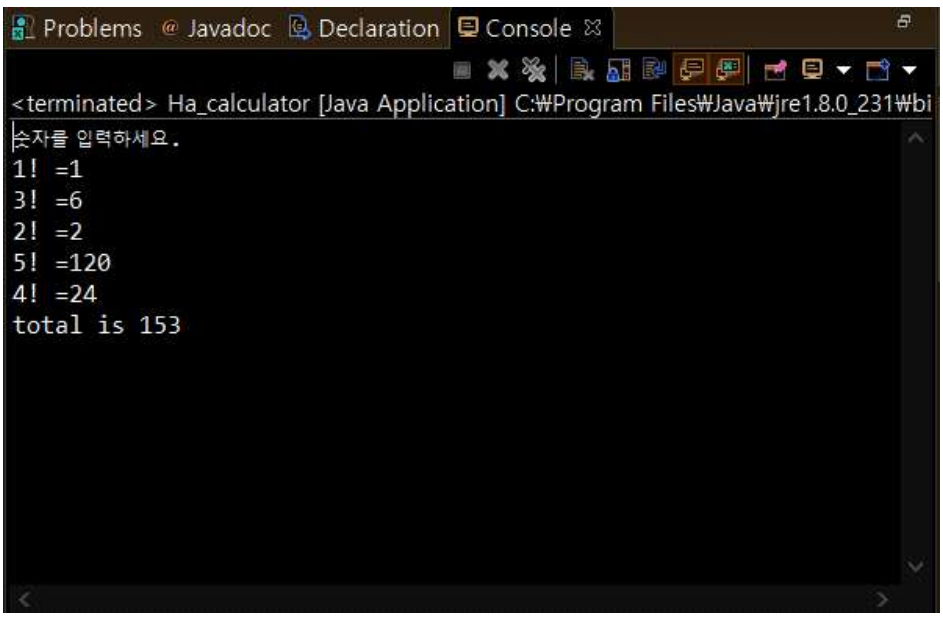
스레드가 전역변수에는 모두 접근 할 수 있기 때문에 sum을 static 변수로 선언했다.

```
if(num == 1) {
                result = 1;
              }
else {
                result = num * factorial(num-1);
     }
return result;
```

factorial을 구하는 메소드이다.

5. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)





(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

작성된 프로그램을 실행한 결과 위와 같은 화면이 출력 되었다.

6. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 저번 과제와 다르게 Runnable 인터페이스를 이용하지 않고, Thread 클래스를 상속받아서 멀티스레드를 구현하였다. 이번 과제는 argument로 숫자 하나를 입력받고, 그 입력받은 수의 factorial 값을 출력하고 그 값들을 다 더해서 출력하는 것이었다. 입력받은 수 만큼 객체를 생성하기 위해서 객체 배열을 이용했다. 객체 배열의 레퍼런스 변수를 먼저 생성한 후, for문을 통해 객체를 생성했다. 또한 factorial값을 계산하고 출력 한 후 그 값을 저장할 때 모든 스레드가 접근할 수 있는 전역변수에 저장했다. 처음에 sum+ = result;를 return result; 윗줄에 작성하였는데 total 값이 제대로 출력되지 않고 값이 더 크게 나왔다. 곰곰이 생각해보니 factorial을 구하는 메소드에서 최종 계산된 result만 sum에 저장되어야 하는데 위처럼 코드를 작성하니까 조금씩 더 더해지는 것이었다. 그래서 sum+ =result;를 각 값의 factorial을 출력하기 전에 작성하니까 제대로 된 값이 나왔다. 그리고 join() 메소드를 이용하여 다른 스레드의 종료를 기다리고 factorial의 sum을 출력했다. 처음에는 스레드가 너무 어렵게 느껴졌지만, 과제를 통해 직접 코딩 해보니 조금씩 이해가 되는 것 같다. 비슷한 예제를 비롯해서 조금 더 많은 문제들을 코딩해봐야겠다.

7. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

```
package HA_homework;


public class Ha_calculator {

	public static void main(String[] args) {
		int key = Integer.parseInt(args[0]);
		System.out.println("숫자를 입력하세요. ");


		MultiThread[] mt = new MultiThread[key]; //입력 받은 숫자만큼 객체 배열의 레퍼런스 변수를 생성한다.
		for(int i=0;i<key;i++) {
			mt[i]=new MultiThread(); //객체 생성
			mt[i].store(i+1); // 1,2,3,4 ... key 까지 순서대로 값을 저장시킨다.
			mt[i].start();
		}
		try {
			for(int i=0;i<key;i++) {
				mt[i].join(); //다른 스레드의 종료를 기다린다.
			}
		}
		catch(InterruptedException e) {
			Thread.currentThread().interrupt(); //스레드를 종료하기 위한 메커니즘
		}
		System.out.println("total is "+ MultiThread.sum); //최종 total 값을 출력한다.

	}
}

class MultiThread extends Thread{
	static int sum=0; //전역변수 생성
	fac_calculator c;
	int num;
	MultiThread(){
		c =new fac_calculator(); //생성자에서 fac_calculator 객체 생성
	}
	public void store(int n) { //각각의 factorial 값을 구하기 위해 숫자를 저장시킨다.
		num = n;
	}
	public void run() {
		try {
			int result = c.factorial(num);
			sum += result; //static 변수 sum에 결과 값을 더해서 저장한다.
			System.out.println(num+"! ="+ result); //각각의 factorial의 합을 출력한다.
		}catch(Exception e) {
			e.printStackTrace(); //에러 메세지의 발생 근원지를 찾아 단계별로 에러 출력
			System.out.println("예외 발생!!");
		}
	}
}
class fac_calculator{
	public synchronized int factorial(int num) { //임계 영역을 설정하여 현재 데이터를 사용하고 있는 해당 스레드를 제외하고,나머지 스레드들은 데이터에 접근할 수 없도록 막기 위한 것
	int result =0;
		if(num == 1) {
		result = 1;
	}
		else {
			result = num * factorial(num-1);
		}
		return result; //factorial의 합을 리턴
	}
}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)