정적 메모리 할당

- 프로그램이 시작되기 전에 "미리 정해진 크기의 메모리를 할당"받는 것을 말함
  - 예, int num; char str[256]; 등
- 처음에 결정된 크기의 값만을 처리함
- int a[1000]; 과 같은 배열은 1000개의 정수값만을 배열에 넣을 수 있으며 1개의 값만이 배열에 있을 경우 메모리 낭비가 발생한다
- 메모리의 'data'영역에 할당됨.

1. 실행 중, 할당된 메모리 크기를 조절할 수 없음.
2. 프로세스가 종료될 때, 그제서야 해당 정적 메모리 할당이 해지됨.

'참조자료형'은 변수에 저장될 때 '동적 메모리 할당'을 실시함.

동적 메모리 할당

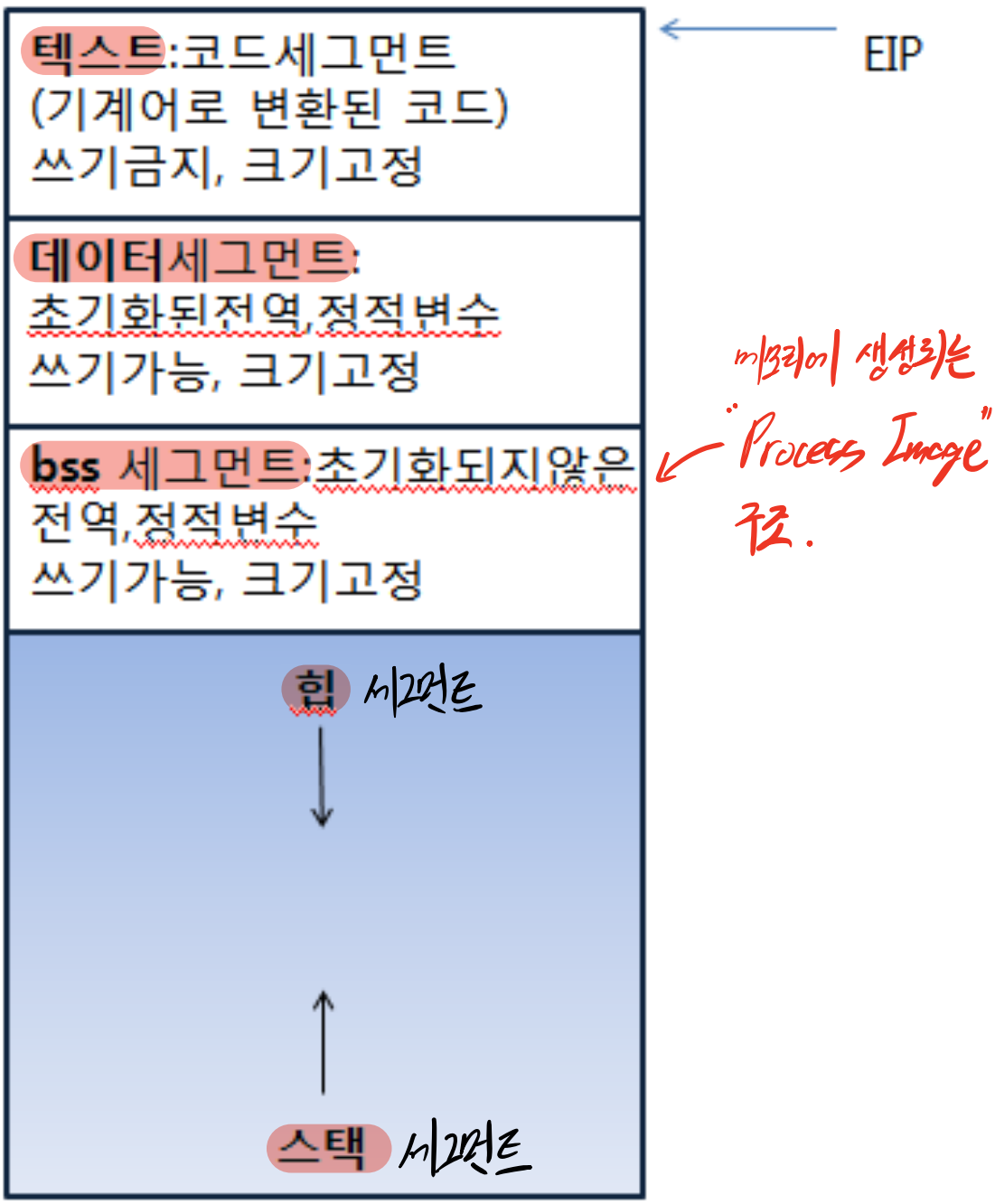
- 프로그램의 실행 도중에 메모리를 할당받는 것
- 운영체제의 힙 영역의 메모리를 할당받아서 사용함
- 메모리 낭비가 적어 효율적임
- 운영체제에 프로그램이 필요로 하는 메모리의 용량을 알려주어야 함. 또한 할당된 메모리의 주소를 알아야 사용할 수 있음
  - C언어는 이를 위해 malloc() 함수를 사용하며, malloc() 함수는 할당된 메모리의 주소값을 반환한다
- 메모리 내 'Heap' 영역에 할당됨.

프로세스 실행 도중에도 할당된 메모리를 해제할 수 있음.(Python의 'del' 키워드)
(프로세스가 종료되면, 자동해제됨).

즉슥, 할당된 메모리 크기를 조절할 수 없음.

· JAVA에선 'new'키워드로 생성한 인스턴스를 저장하는 변수 모두 '동적할당'에 해당함.

※ '메모리 할당'의 의미 : 자료형 크기만큼의 공간을 메모리 내에서 확보하는 것.



텍스트 세그먼트(코드 세그먼트)
이 영역은 작성한 코드가 기계어로 바뀌어 저장 됩니다. EIP는 이 코드의 흐름을 읽는 EIP레지스터입니다. 이 곳에는 변수가 아닌 순수 코드만 있는 영역입니다.(함수, 제어문 등등..)
함수의 주소값이 4200으로 시작하는군요. 우선 아래 영역들의 주소값을 알아보고 비교하겠습니다.

데이터 세그먼트
초기화된 전역&정적 변수가 저장되는 곳입니다. 위 코드의 실행 결과 화면에서 초기화된 전역, 정적변수의 주소를 찾아보면
b1,b2, d1,d2,d3 입니다. 주소가 4210로 시작합니다. 동일한 세그먼트에 저장됨을 알 수 있습니다.

bss 세그먼트
초기화 되지 않은 정적,전역변수는 모두 4223으로 시작하는 주소값을 가졌습니다.

① 여기까지 주소값의 영역을 정리하면 4200xxx->4210xxx->4223xxx (코드->데이타->bss) 로 위 그림과 동일함을 알 수 있습니다.
② 또한, 변수 선언 순서에 따라 주소값이 커지는 구조입니다.

"정적변수"가 저장되는 곳 "지역변수"가 저장되는 곳

힙과 스택 세그먼트
동적할당을 하여 힙 영역을 살펴본 결과 여기서는 6으로 시작하는 주소값을 가졌습니다.할당 순서에 따라서 주소값이 커지네요.
그런데 스택영역은 2로 시작합니다.
위 그림과는 일치하지는 않습니다. 단지 스택이 높은 주소에서 낮은 주소로 쌓여가는 것은 맞네요.
제 짧은 지식으로는 잘 모르겠군요. 추측하면, 메모리 모델의 차이라고 생각합니다.
사실 일반적인 프로그래밍을 하는 경우 이런 부분이 크게 필요한 부분은 아니기 때문에, 그러려니 하고 넘어가겠습니다.

const 선언시
모든 전역 const 변수는 데이타 세그먼트에 저장됩니다.
지역 const 변수일 경우 문자열은 데이타 세그먼트에 저장되고, int 형 변수(double...도 마찬가지)는 스택에 저장됩니다.

Java에서 변수를 사용할 때, 원시 데이터 타입(Primitive type)이 아닌 이상 모든 데이터 타입은 new로 할당이 됩니다.
예를 들어, 문자열의 타입 String을 사용할 때 우리는 String a = "hello world" 라고 사용하기는 하지만 내부적으로는 String a = new String("hello world");와 같습니다.
즉, Java에서는 원시 데이터 타입을 제외하고는 모든 객체가 클래스 타입입니다.

↳ 기본자료형 ↳ 참조자료형

클래스를 사용할 때는 우리가 new라는 객체를 사용합니다.

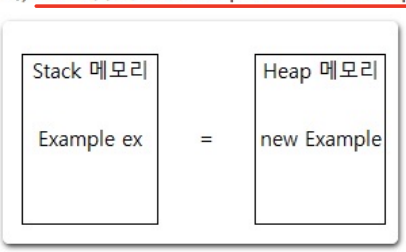
[소스 보기] Example.java

```java
// 클래스
public class Example {
  // 맴버 변수
  private int data;
  // 생성자
  public Example(int data) {
    // 데이터
    this.data = data;
  }
  // 출력 함수
  public void print() {
    // 콘솔 출력
    System.out.println("data - "+ this.data);
  }
  // 실행 함수
  public static void main(String... args) {
    // 클래스 선언
    Example ex = new Example(10);
    // 출력
    ex.print();
  }
}
```

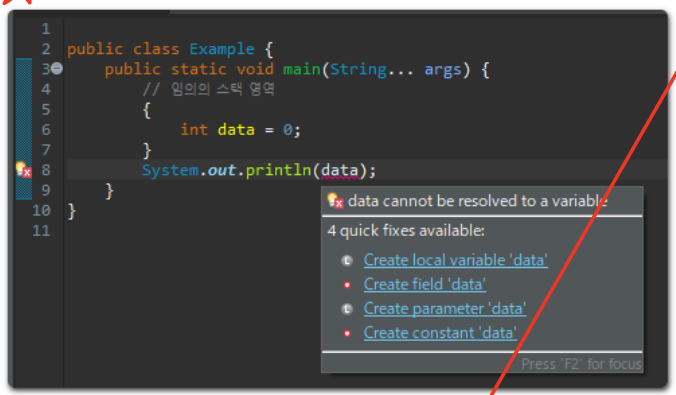
↳ 인스턴스 생성

```
data - 10
```

위 예제는 당연한 것입니다만, 우리가 봐야 할 부분은 Example ex = new Example(10); 입니다.
먼저 앞의 Example ex은 변수 선언입니다. 여기서 우리가 Example ex = 10;이라고 하면 당연히 에러가 발생합니다. 왜냐하면 Example ex에는 Example의 클래스만 할당이 되어야 하기 때문입니다.
즉, Java 내부에서는 Example ex = new Example(10);가 어떤 형태로 생성이 되냐하면 아래의 그림과 같이 생성이 됩니다.



여기서 Stack 메모리와 Heap 메모리에 대한 구조가 나왔습니다.
잠깐 이 Stack 메모리와 Heap 메모리에 대해 이해할 필요가 있습니다.
Stack 메모리는 우리가 프로그램에서 함수를 작성할 때, 실행하는 영역을 중괄호({ })로 설정합니다. 이 중괄호의 영역을 우리는 Stack 영역이라고 이야기합니다. 이 Stack 영역에서 선언된 변수의 값은 우리가 Stack 메모리에 보관 된다고 합니다.



위 이미지를 보시면 main 함수 안에 임의의 중괄호를 사용해서 새로운 스택 영역을 만들었습니다. 그 새로운 스택 영역에서 선언된 data는 스택 영역을 벗어나서는 사용할 수 없습니다.
다시 돌아와서 Example ex는 스택 영역에서 선언된 것입니다.

new Example(10)는 Heap 영역에서 할당된 것인데, Heap은 프로그램의 영역입니다. 즉, 프로그램이 실행되면서 Heap 메모리가 생성이 되고 그 안에서 자유롭게 선언하고 해지를 할 수 있습니다.
그러나 이 Heap은 딱히 어디에서 선언되고 메모리에 어디에 박혀있는지 알 수 없습니다. 그래서 Heap에서 선언된 new Example의 주소 값을 Stack메모리에 선언된 Example ex에 넣는 것입니다.

정리하면 Stack은 정적인 메모리 영역이며 데이터를 찾기가 쉽지만(Stack 알고리즘의 push pop으로 데이터를 찾는다.), Heap은 동적인 메모리 영역이며 new키워드로 클래스를 할당하면 데이터를 주소 값으로만 찾을 수 있습니다.
그리고 이 둘을 연결한 게 Example ex = new Example(10);의 형태입니다.

함수를 호출 시 함수의 매개변수, 지역변수, 리턴 값, 그리고 함수 종료 후 돌아가는 위치가 스택 메모리에 저장된다.

재귀함수를 쓰게되면, 함수를 반복적으로 호출하므로, 스택 메모리가 커지고, 호출하는 횟수가 많아지면 스택오버플로우가 발생할 수 있다.