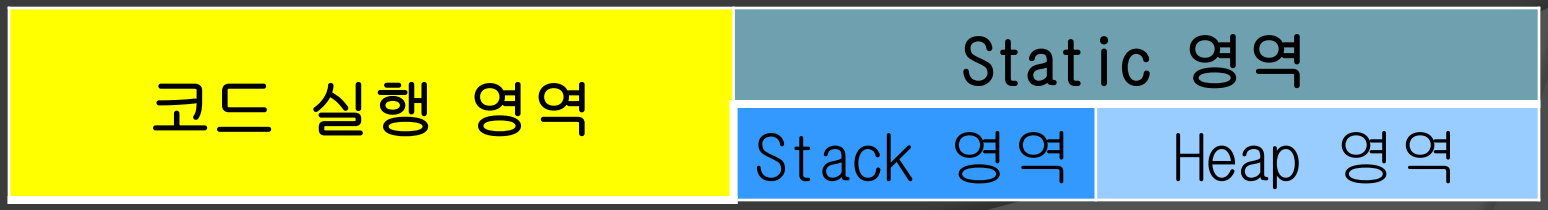
JAVA

메모리 구조

프로그램이 메모리를 사용하는 방식

하나의 프로그램이 실행될 때 프로그램이 메모리를 사용하는 방식
기계어를 포함한 모든 프로그래밍 언어의 공통된 메모리 사용 방식

객체 지향 프로그램에서는 데이터 저장 영역을 다시 3개의 영역으로 분할해서 사용한다.



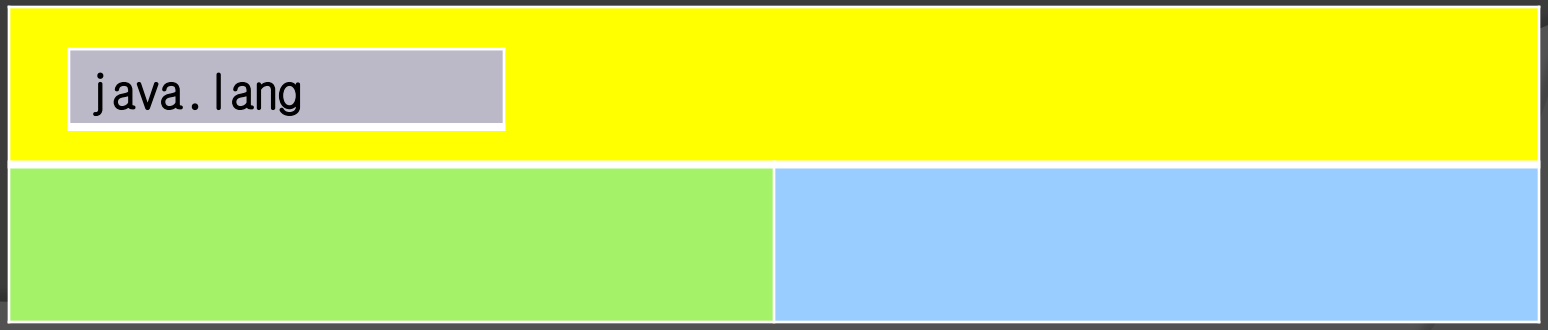
프로그램이 메모리를 사용하는 방식

```
public class Start1 {
  public static void main(String[] args) {
    System.out.println( “Hello Memory!” );
  }
}
```

Static 영역 - Class	
Stack 영역 - Method	Heap 영역 - 객체

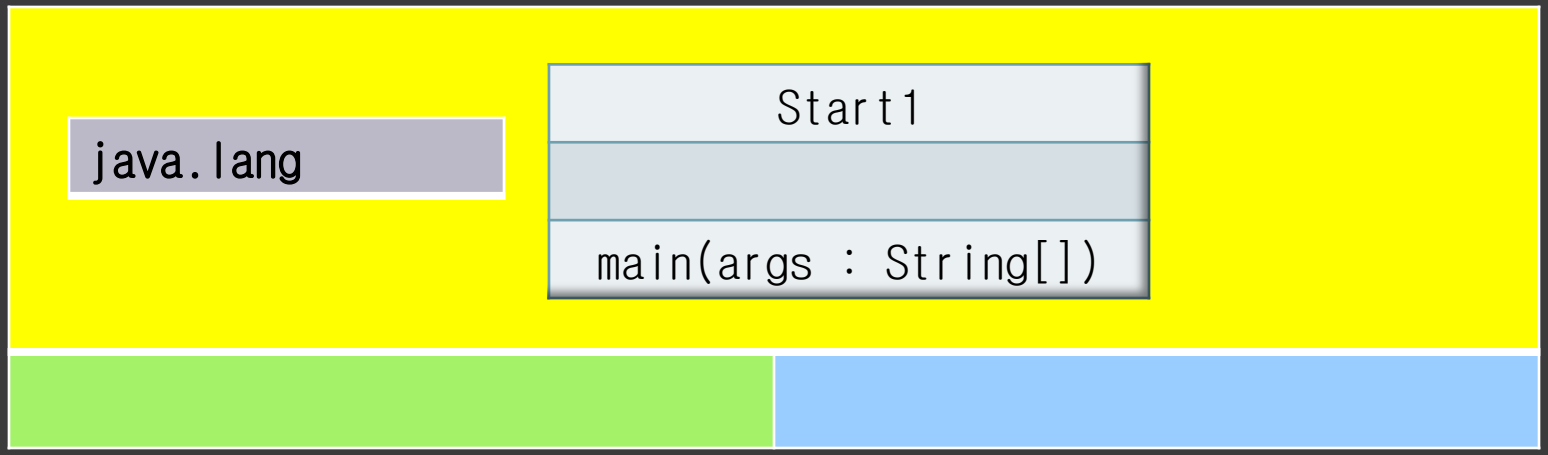
프로그램이 메모리를 사용하는 방식

JRE는 먼저 프로그램 안에 main() 메서드가 있는지 확인한다.
main() 메서드의 존재가 확인되면 JRE는 프로그램 실행을 위한 사전 준비에 착수한다.
가상의 기계인 JVM에 전원을 넣어 부팅하는 것이다.
부팅된 JVM은 목적 파일을 받아 그 목적 파일을 실행한다.
JVM이 맨 먼저 하는 일은 전처리하고 하는 과정이다.
모든 자바 프로그램이 반드시 포함하게 되는 패키지가 있다.
  - java.lang 패키지
JVM은 가장 먼저 java.lang 패키지를 T메모리의 static영역에 가져다 놓는다.
java.lang 패키지가 있기에 System.out.println() 같은 메서드를 쓸 수 있는 것이다.



프로그램이 메모리를 사용하는 방식

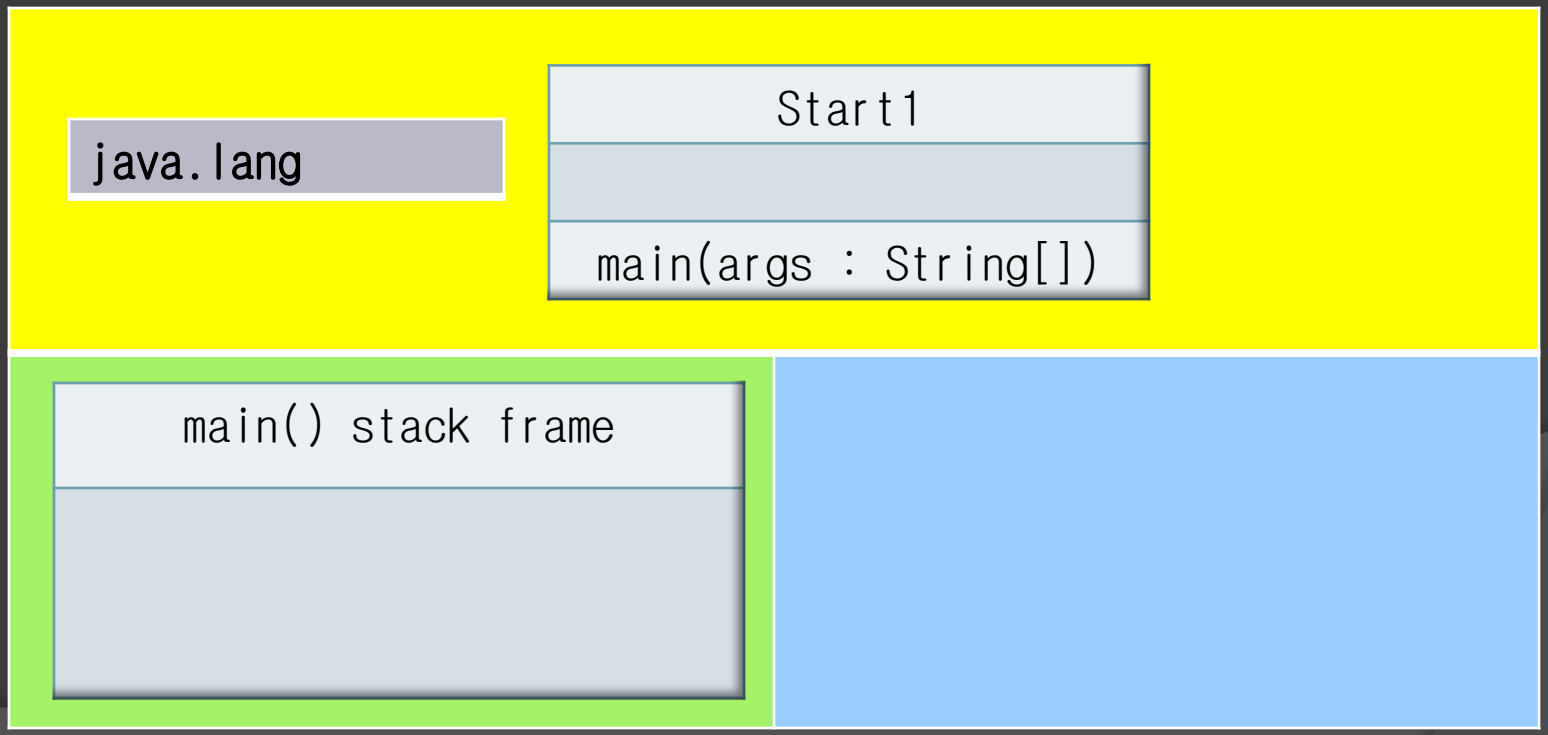
JVM은 개발자가 작성한 모든 클래스와 임포트 패키지 역시 static 영역에 가져다 놓는다.



main() 메서드가 실행되기 전 JVM에서 수행하는 전처리 작업들
-java.lang 패키지를 T메모리의 static 영역에 배치한다.
-import된 패키지를 T메모리의 static 영역에 배치한다.
-프로그램 상의 모든 클래스를 T메모리의 static 영역에 배치한다.

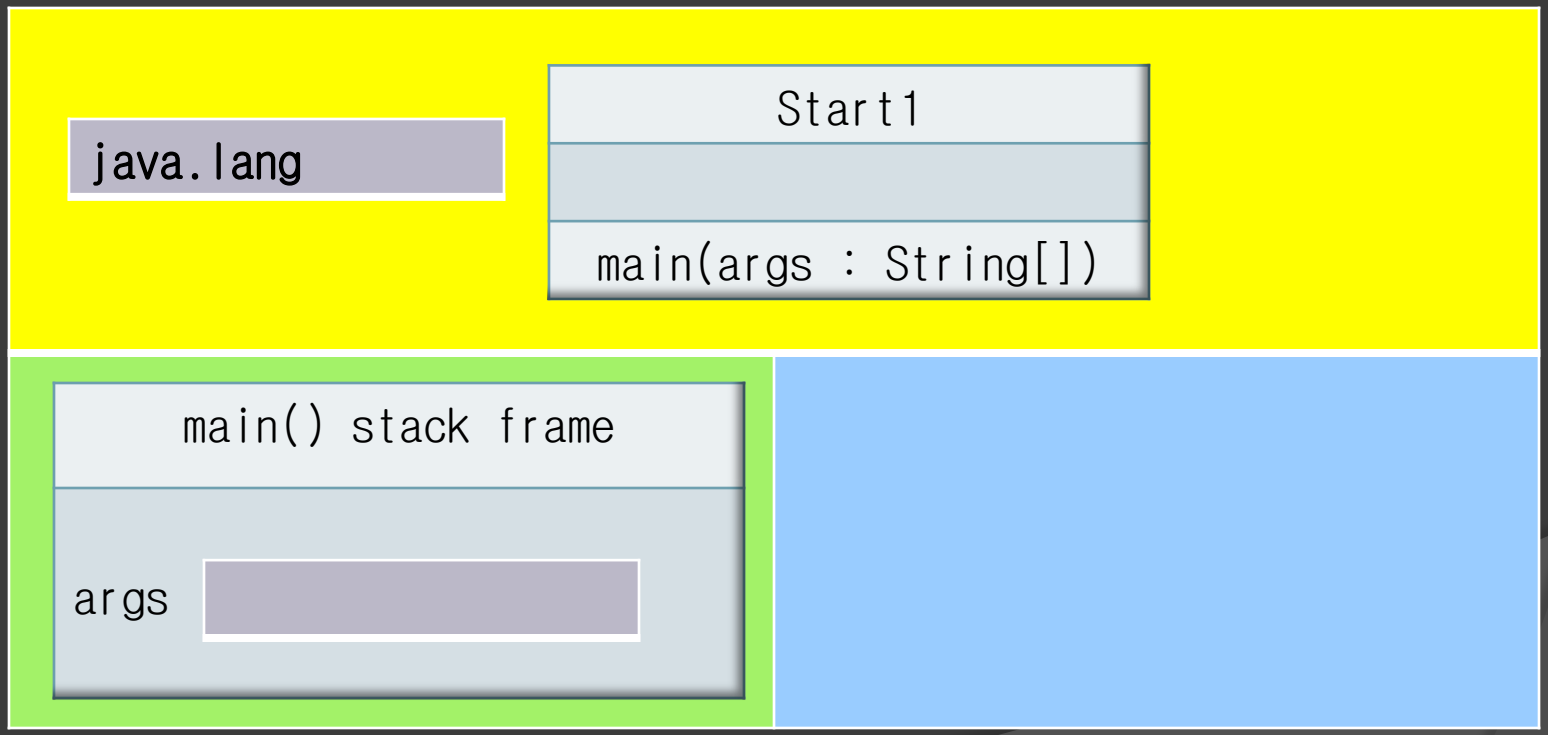
프로그램이 메모리를 사용하는 방식

아직 System.out.println(“Hello Memory!”);이 실행되지 않는다.
main() 메서드내의 문장들을 실행하기 위해 stack frame이 stack 영역에 할당된다.
열리는 중괄호를 만날 때 마다 stack frame이 하나씩 생긴다.
클래스 정의를 시작하는 여는 중괄호만 빼고



프로그램이 메모리를 사용하는 방식

아직 System.out.println(“Hello Memory!”);이 실행되지 않는다.
메서드의 인자 args를 저장할 변수 공간을 stack frame의 맨 밑에 확보해야 한다.
즉, 메서드 인자(들)의 변수 공간을 할당하는 것이다.



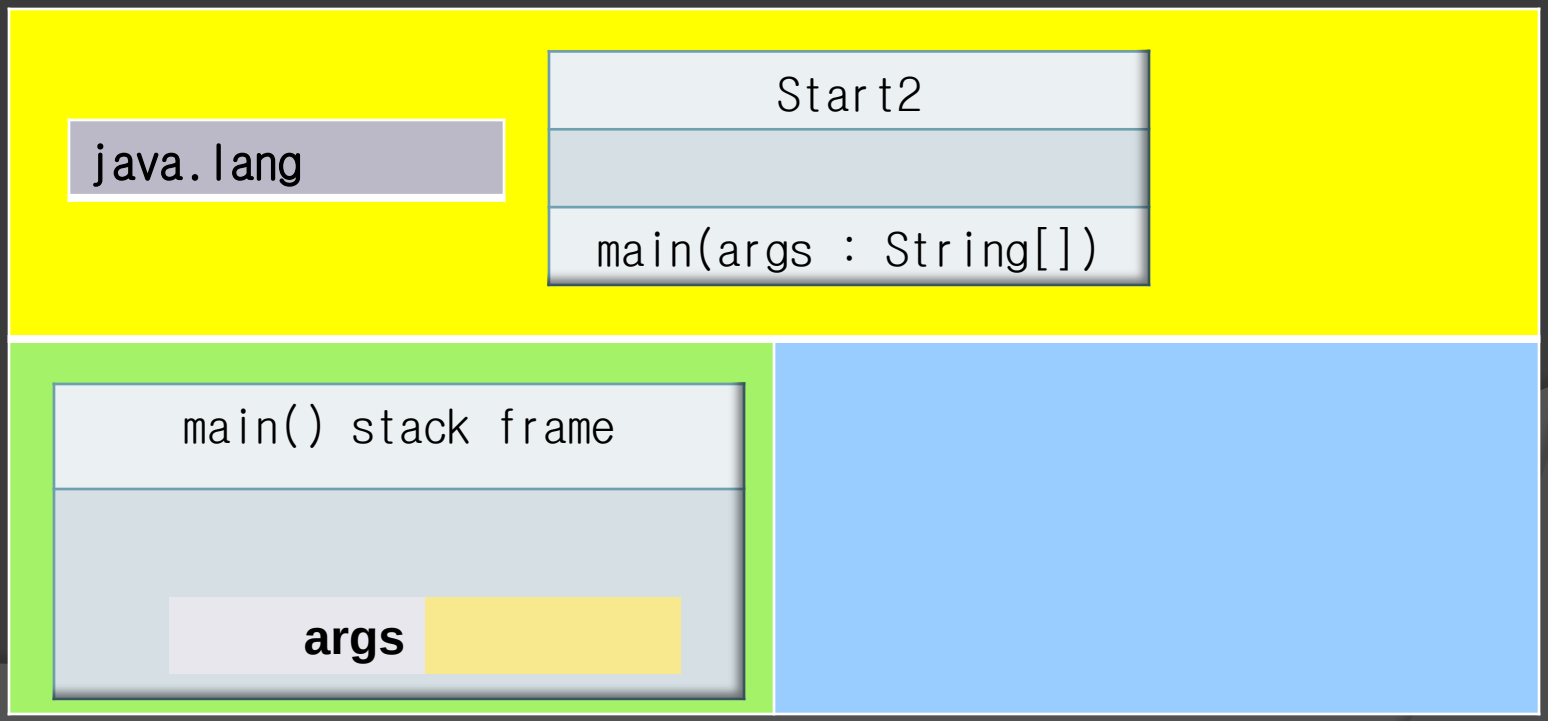
이제 main() 메서드 안의 첫 명령문을 실행하게 된다.

프로그램이 메모리를 사용하는 방식

JRE는 눈에 보이지 않게 뒤에서 JVM이라고 하는 자바 가상 기계를 부팅하고, JVM은 메모리 구조를 만들고 거기에 java.lang패키지 로딩, 각종 클래스 로딩, main() 메서드 스택 프레임 배치, 변수 공간 배치 등의 일을 처리한다.

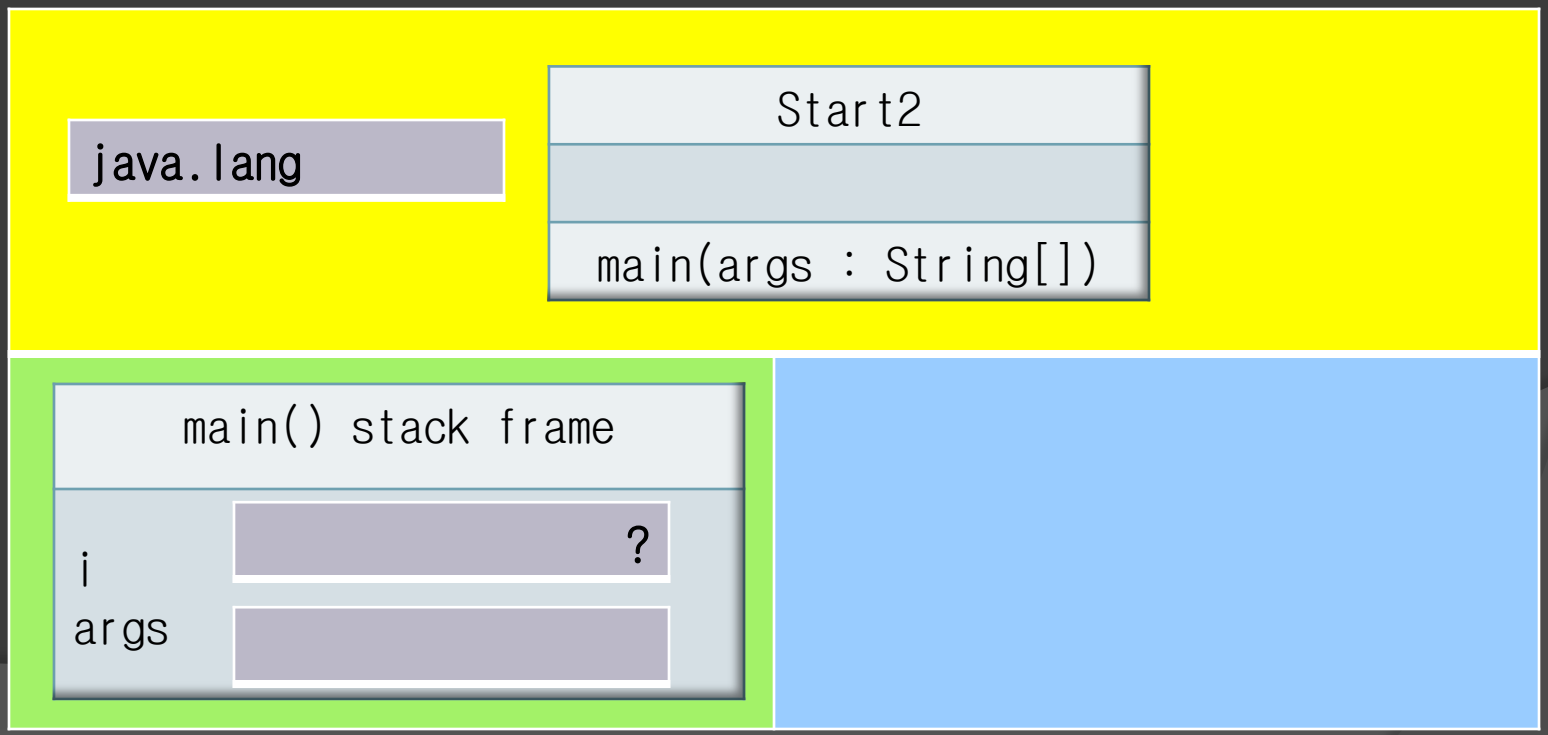
프로그램이 메모리를 사용하는 방식

```
public class Start2 {
  public static void main(String[] args) {
    int i;
    i = 10;
    double d = 20.0;
  }
}
```



프로그램이 메모리를 사용하는 방식

int i; => 메모리에 4byte 크기의 정수 저장 공간을 마련하라.
JVM은 i변수를 위한 공간을 마련한다.
i는 main() 메서드 내에 있으니 main() 메서드 스택 프레임 안에 밑에서부터 변수공간을 마련한다.



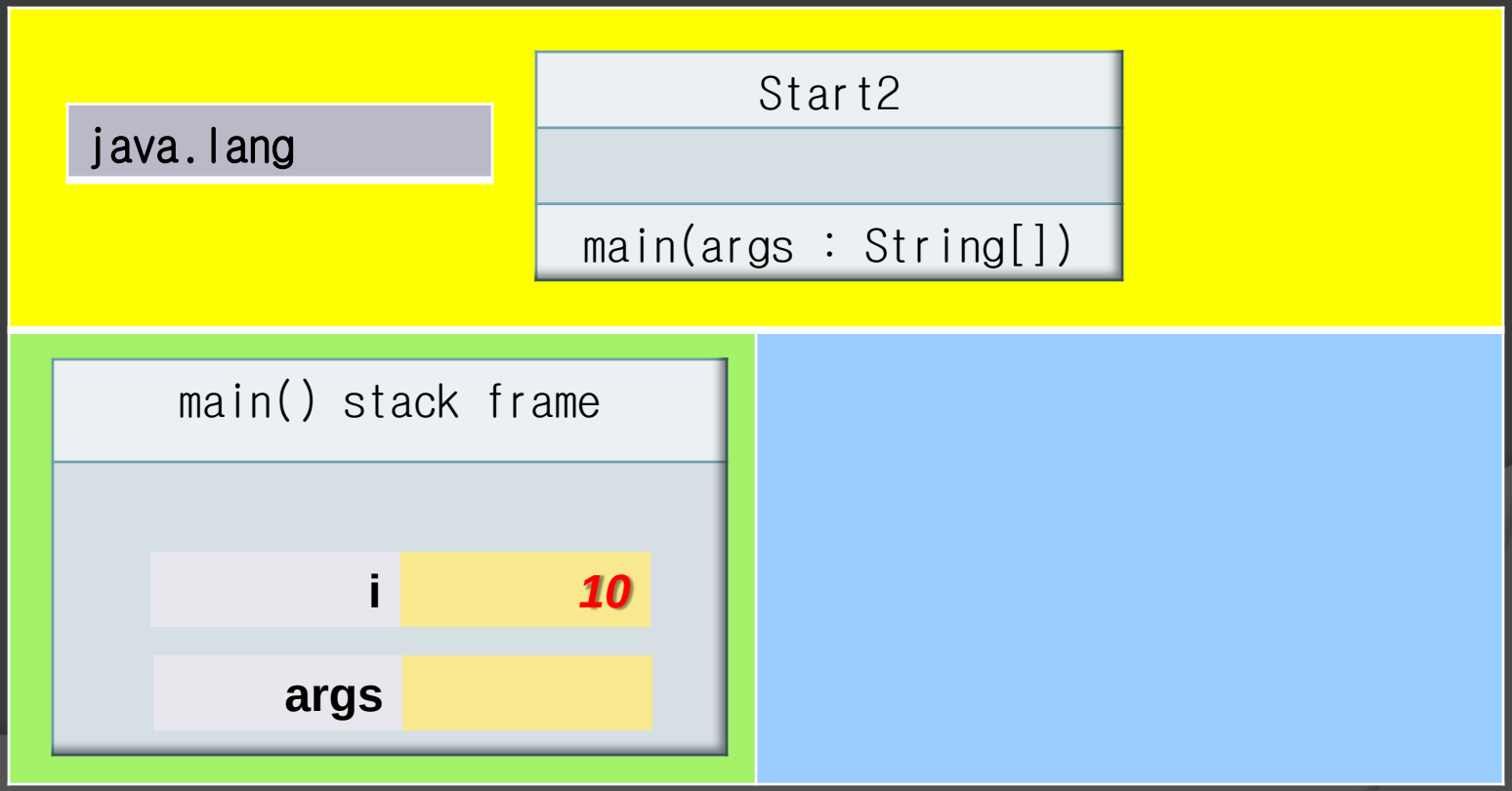
프로그램이 메모리를 사용하는 방식

현재 변수 i에 저장된 값은 얼마일까?
물음표(?)에는 무슨 값이 들어 있을까?
알수 없는 값이 들어가 있다.
이전에 해당 공간의 메모리를 사용했던 다른 프로그램이 청소하지 않고 간 값을 그대로 가지고 있다.

변수 i를 선언만 하고 초기화하지 않은 상태에서 i변수를 사용하는 코드를 만나면 javac(자바 컴파일러)는
"The local variable i may not have been initialized" 경고를 출력한다.(그 지역 변수 i는 초기화되지 않았을 수도 있습니다.)

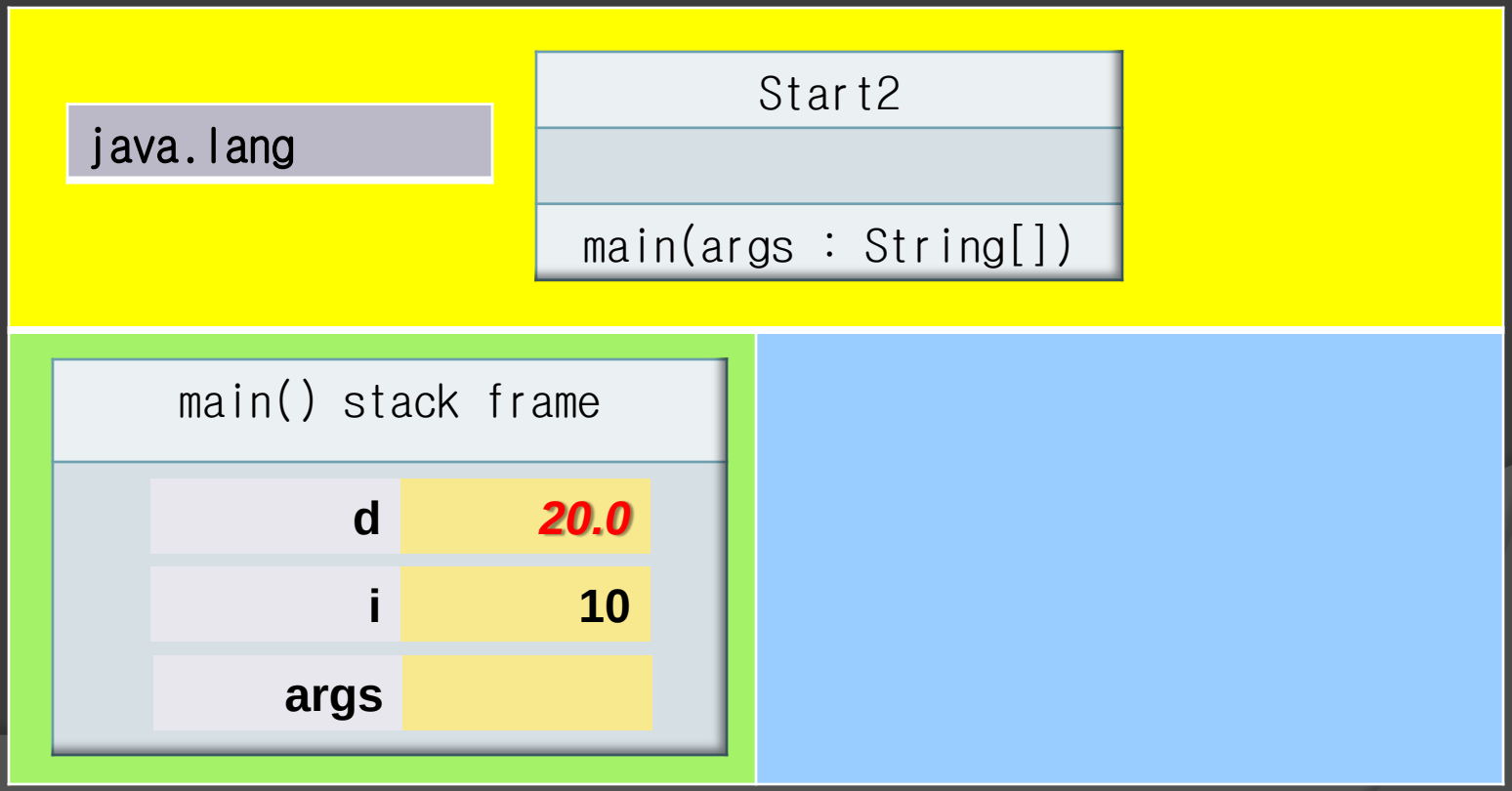
프로그램이 메모리를 사용하는 방식

i = 10; =>



프로그램이 메모리를 사용하는 방식

double d = 20.0; =>

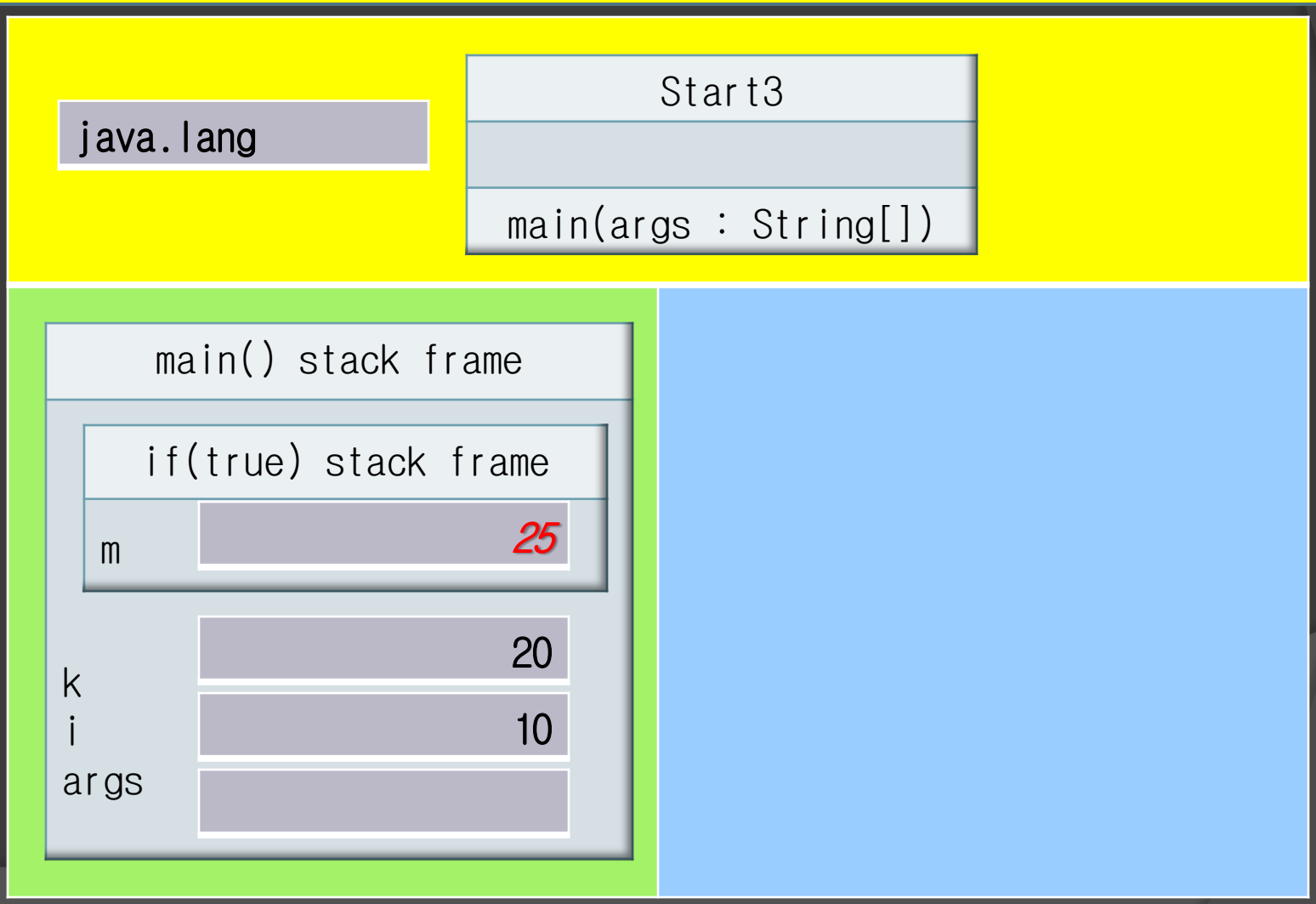


프로그램이 메모리를 사용하는 방식

```
public class Start3 {
  public static void main(String[] args) {
    int i = 10;
    int k = 20;

    if(i == 10) {
      int m = k + 5;
      k = m;
    } else {
      int p = k + 10;
      k = p;
    }
  }
}
```

프로그램이 메모리를 사용하는 방식

int m; => m = k + 5;



프로그램이 메모리를 사용하는 방식

- T메모리는 구역이 3군데 있는데 변수는 static영역, stack영역, heap영역 중 어디에 있는 걸까?
- 3군데 모두이다. 3군데 각각에 있는 변수는 각기 다른 목적을 가진다.
- 각각의 이름도 지역 변수, 클래스 멤버 변수, 객체 멤버 변수로 다르다.

- 지역 변수는 stack 영역에서 일생을 보내게 된다.
- 그것도 stack frame 안에서 일생을 보내게 된다.
- 따라서 stack frame이 사라지면 함께 사라진다.

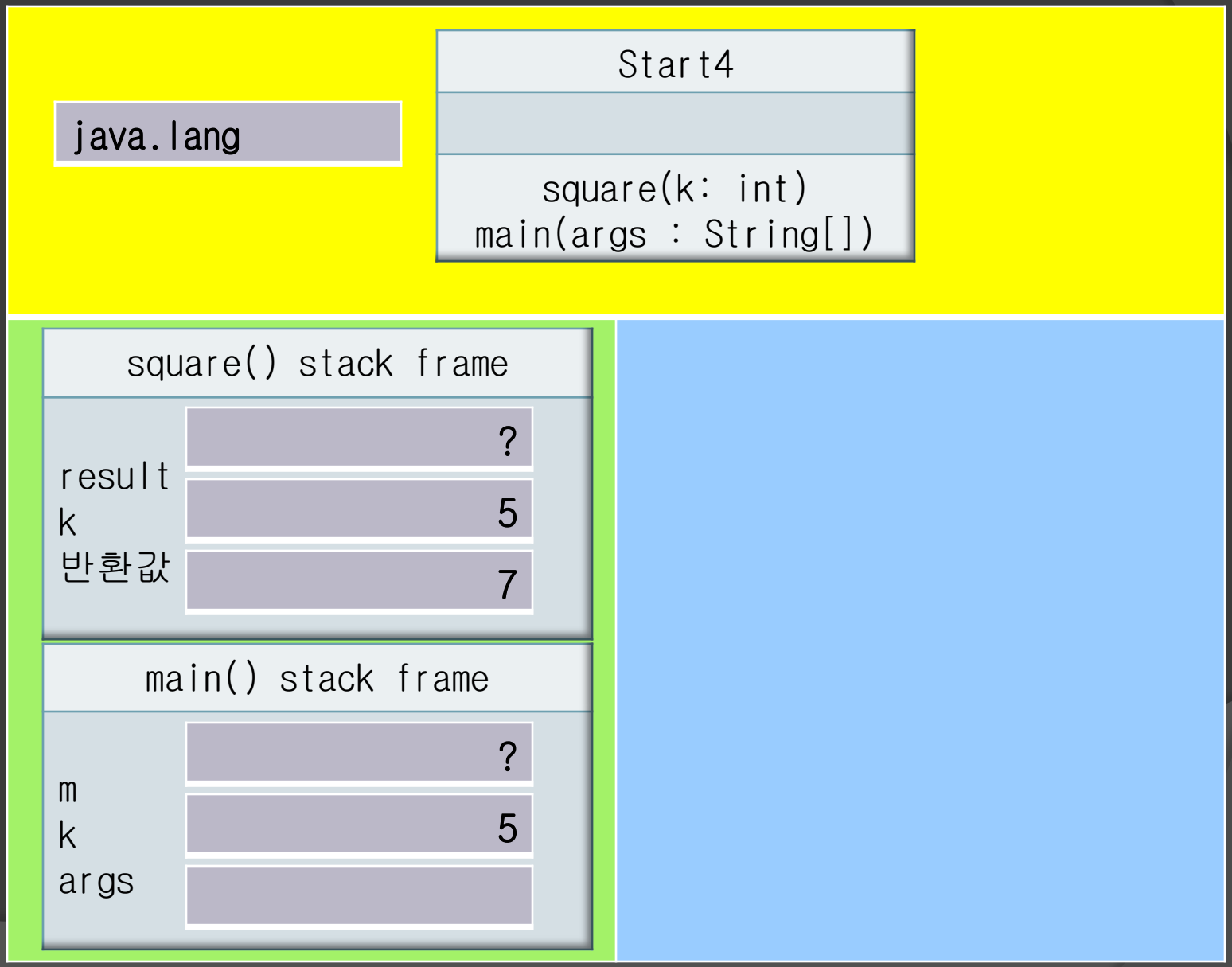
- 클래스 멤버 변수는 static 영역에서 일생을 보내게 된다.
- static 영역에 한번 자리 잡으면 JVM이 종료될 때까지 고정된(static) 상태로 그 자리를 지킨다.

- 객체 멤버 변수는 heap 영역에서 일생을 보내게 된다.
- 객체 멤버 변수는 객체와 함께 가비지 컬렉터라고 하는 heap 메모리 회수기에 의해 일생을 마치게 된다.

프로그램이 메모리를 사용하는 방식

```
public class Start4 {
  public static void main(String[] args) {
    int k = 5;
    int m;

    square(k);
  }
  private static int square(int k) {
      int result;
      k = 25;
      result = k;
      return result;
  }
}
```

프로그램이 메모리를 사용하는 방식



전역 변수와 메모리 : 전역 변수 쓰지 마세요

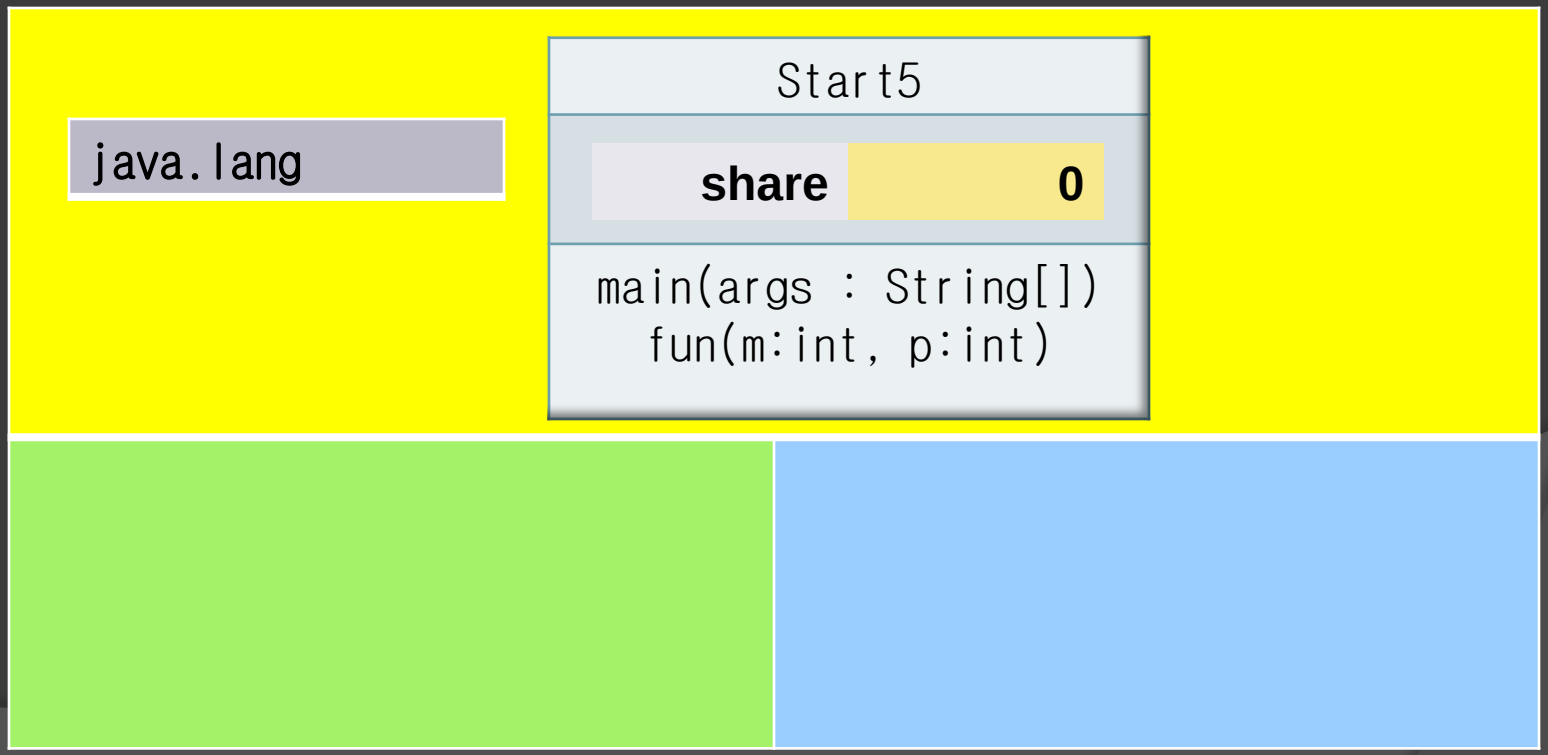
- 두 메서드 사이에 값을 전달하는 방법은
- 1.메서드를 호출할 때 메서드의 인자를 이용하는 방법
- 2.메서드를 종료할 때 반환값을 넘겨주는 방법
- 3.전역 변수를 사용하는 방법

프로그램이 메모리를 사용하는 방식

```
public class Start5 {
  static int share;

  public static void main(String[] args) {
    share = 55;
    int k = fun(5, 7);

    System.out.println(share);
  }
  private static int fun(int m, int p) {
      share = m + p;

      return m - p;
  }
}
```

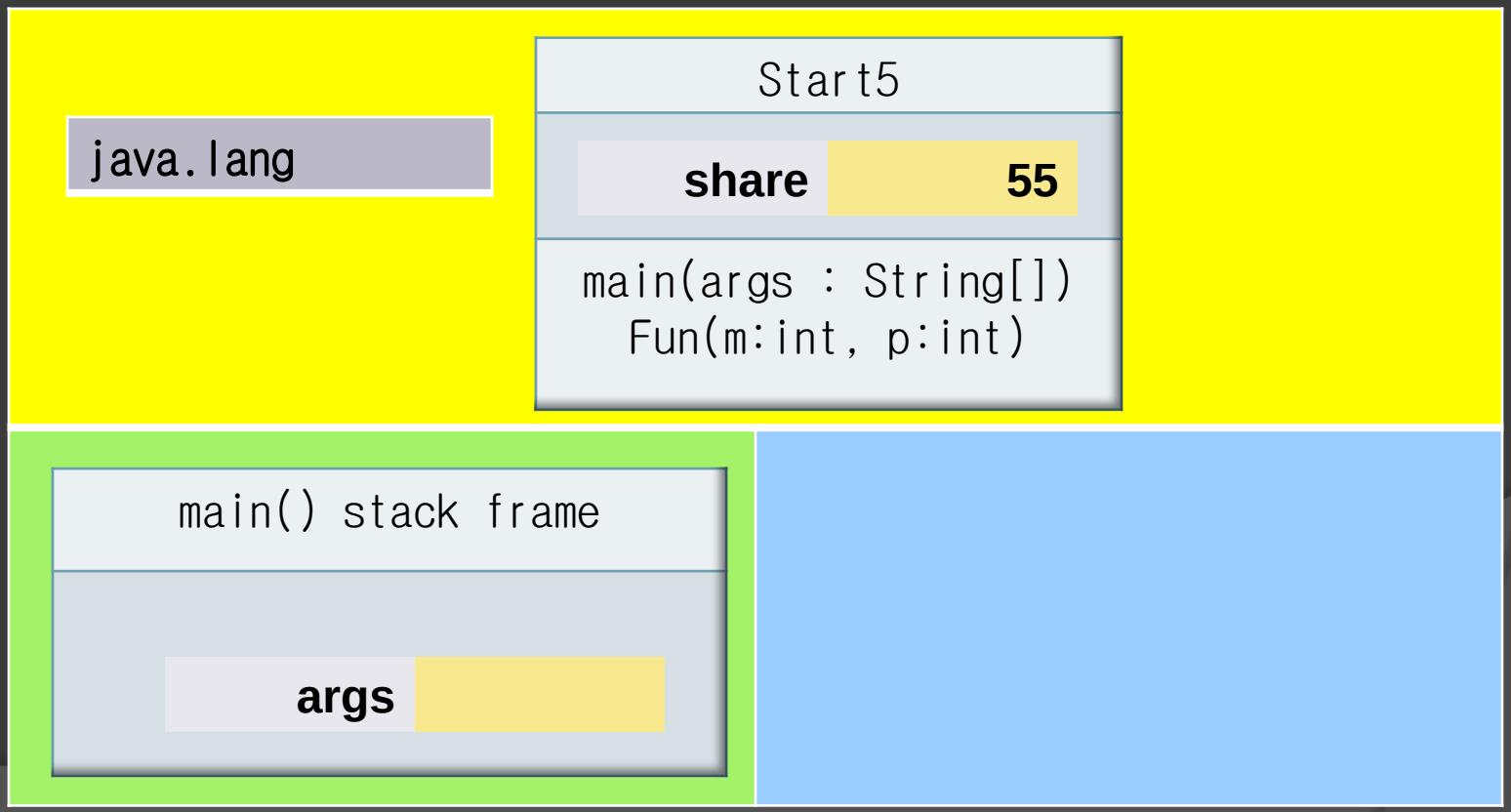
프로그램이 메모리를 사용하는 방식

Share 변수에 static 키워드가 붙어있다.
그래서 share 변수는 T메모리의 static 영역에 변수 공간이 할당된다. 그것도 Start5 클래스 안에 정의되었으니 해당 클래스가 T메모리 static 영역에 배치될 때 그 안에 share 변수가 클래스의 멤버로 공간을 만들어 저장된다.



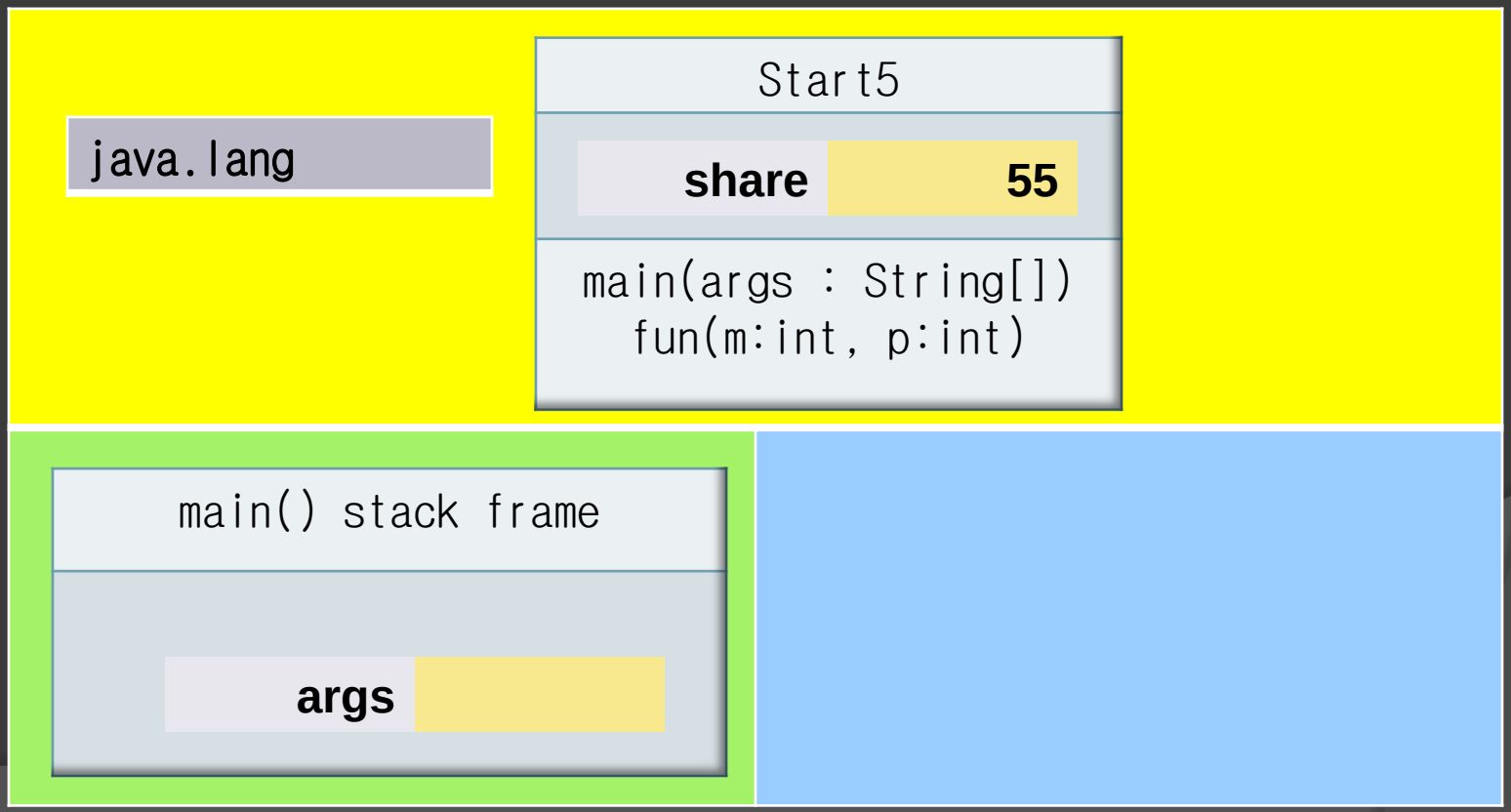
프로그램이 메모리를 사용하는 방식

public static void main(String[] args) { => 이 줄이 실행되면 T
메모리에 main() 메서드 스택 프레임이 만들어진다.
share = 55;



프로그램이 메모리를 사용하는 방식

int k = fun(5, 7); => 이 줄을 실행하기 위해 main() 메서드 스택 프레임에 k변수 공간이 만들어진다. 이어서 제어는 fun() 메서드가 있는 private static int fun(int m, int p) { 로 넘어간다. 그러면 fun() 메서드 스택 프레임이 생성되고 인자값들과 변환값을 저장할 변수 공간도 생긴다.



프로그램이 메모리를 사용하는 방식

프로그램이 메모리를 사용하는 방식

share = m + p; 가 실행되고 나면 m 변수와 p 변수에 있는 값을 더해 share 변수에 할당한 상태가 된다.

프로그램이 메모리를 사용하는 방식

프로그램이 메모리를 사용하는 방식

return m - p; 가 실행되고 나면

프로그램이 메모리를 사용하는 방식

- 전역 변수는 코드 어느 곳에서나 접근할 수 있다고 해서 전역 변수라고 하며,
- 여러 메서드들이 공유해서 사용한다고 해서 공유 변수라고도 한다.

- 그런데 전역 변수를 쓰지 말라고들 하는데 왜 그럴까?

- 프로젝트 규모에 따라 코드가 커지면서 여러 메서드에서 전역 변수의 값을 변경하기 시작하면 전역 변수에 저장되어 있는 값을 파악하기 쉽지 않기 때문이다.

- 수천 줄이 넘는 코드에서 다른 메서드에 의해 전역 변수 share에 다른 값이 저장된다면 코드를 추적해 들어가야만 그 값과 그 값이 변한 이유를 파악할 수 있다.

- 결론적으로 가능한 전역 변수는 피하는 것이 좋다.
- 다만 읽기 전용으로 값을 공유해서 전역 상수로 쓰는 것을 추천한다.