CMSC131

Variables in Java

The Java Memory Model and Initializations

The Java Memory Model

There are a ***wide variety*** of things to explore to learn about the entire Java Memory Model.

In this course we have been and will continue to focus on:

- primitive variables, references to objects, and objects
- stack versus heap versus permgen
- single-threaded programs

Stack - Heap - PermGen

The stack and the heap and permgen are three logical areas within a computer's memory where information is stored by a Java Virtual Machine.

They each have a different purpose in a typical Java program.

Stack

The stack can be seen as the part of memory where method parameters and local variables (both primitives and references) are stored when a method is invoked and while it runs.

As each method is invoked a new “stack frame” is created. As each method ends, its frame is “popped” off the stack.

Heap

The heap can be seen as the part of memory where objects live.

Unlike the stack, things added to the heap don’t have to be removed in reverse order and can persist and be accessed beyond the end of a method call as long as references to them are still available.

PermGen

In an update to the Java Memory Model in 2004, the “Permanent Generation” area of memory was added to the two existing areas (stack and heap).

It was decided that certain things, such as static class variables, would be stored there. These static class variables would have space allocated for them as soon as the JVM knew that the program would use their class.

Student class example (*partial*)

file: Student.java

```
public class Student {
      private String name;
      private String ident;
      public int tokenLevel;
      private static int currentCount = 0;
      private static int overallCount = 0;

     public Student(String nameIn, int tokensIn, int identIn) {
         name = nameIn;
         tokenLevel = tokensIn;
         ident = convertIntToString(identIn);
         currentCount++;
         overallCount++;
      }

      .......
      .......

}
```

`Student` class in use…

Let's consider the following lines of code as if it were in the main method of our program and perform a trace of what is placed on the stack, what is placed on the heap, and what is placed in permgen.

```
int val = 7;
Student s1 = new Student("Pat", val, 987654321);
Student s2 = s1;
```

Garbage Collection

The short version is that the parts of memory on the heap that are no longer referenced by anything that can be reached are available to be released for reuse.

Some languages require the programmer to explicitly mark objects for deletion. Java chooses to manage deletion on its own to reduce memory leaks.

Consider the previous picture modified to reflect the additional line of code shown below:

```
int val = 7;
Student s1 = new Student("Pat", val, 987654321);
Student s2 = s1;
s1 = new Student("Sam", 3, 436389456);
```

Is anything on the heap ready for garbage collection?

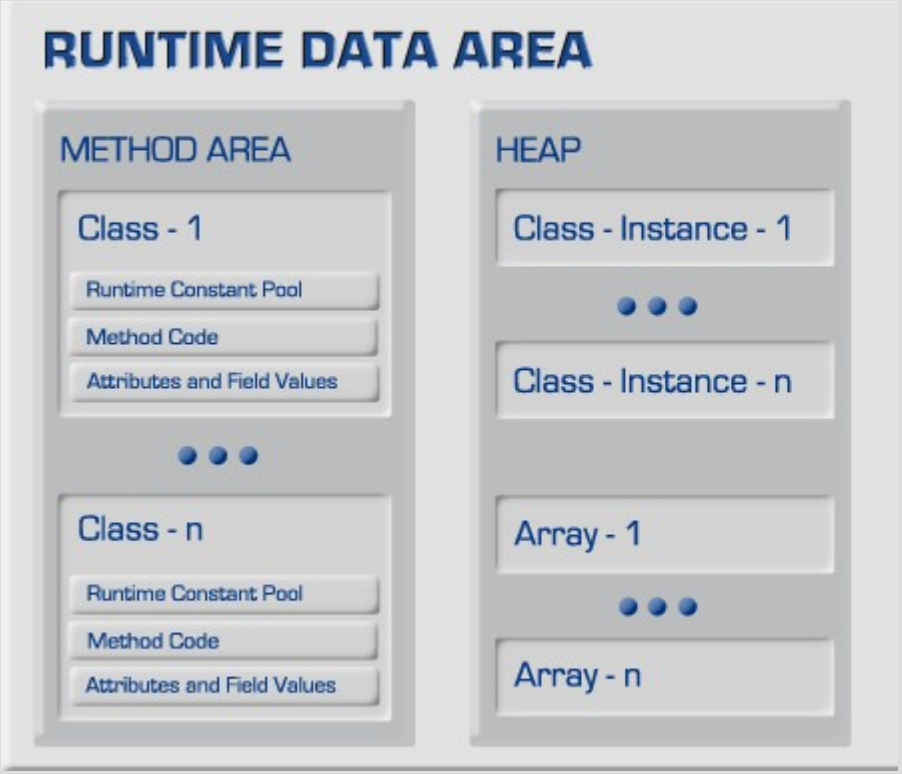
1. Yes
2. No

Consider the previous picture modified to reflect the additional line of code shown below:

```
int val = 7;
Student s1 = new Student("Pat", val, 987654321);
Student s2 = s1;
s1 = new Student("Sam", 3, 436389456);
s2 = new Student("George", 5, 965456746);
```

Is anything on the heap ready for garbage collection?

1. Yes
2. No



http://blog.codecentric.de/wp-content/uploads/2009/12/java-memory-architecture.jpg