سوال اول:

الف) wapper class: رپرکلاس‌ها در واقع راهکار ما در جاوا برای استفاده از دیتاهای ازنوع اصلی(primitive data type) به عنوان آبجکت هستند در جاهایی که ما نمی‌توانیم از این نوع داده‌ها استفاده کنیم. از مثال‌های کاربرد آن کالکشن‌هایی که با آبجکت‌ها کار می‌کنند مانند ArrayList و HashMap و... می‌باشد.

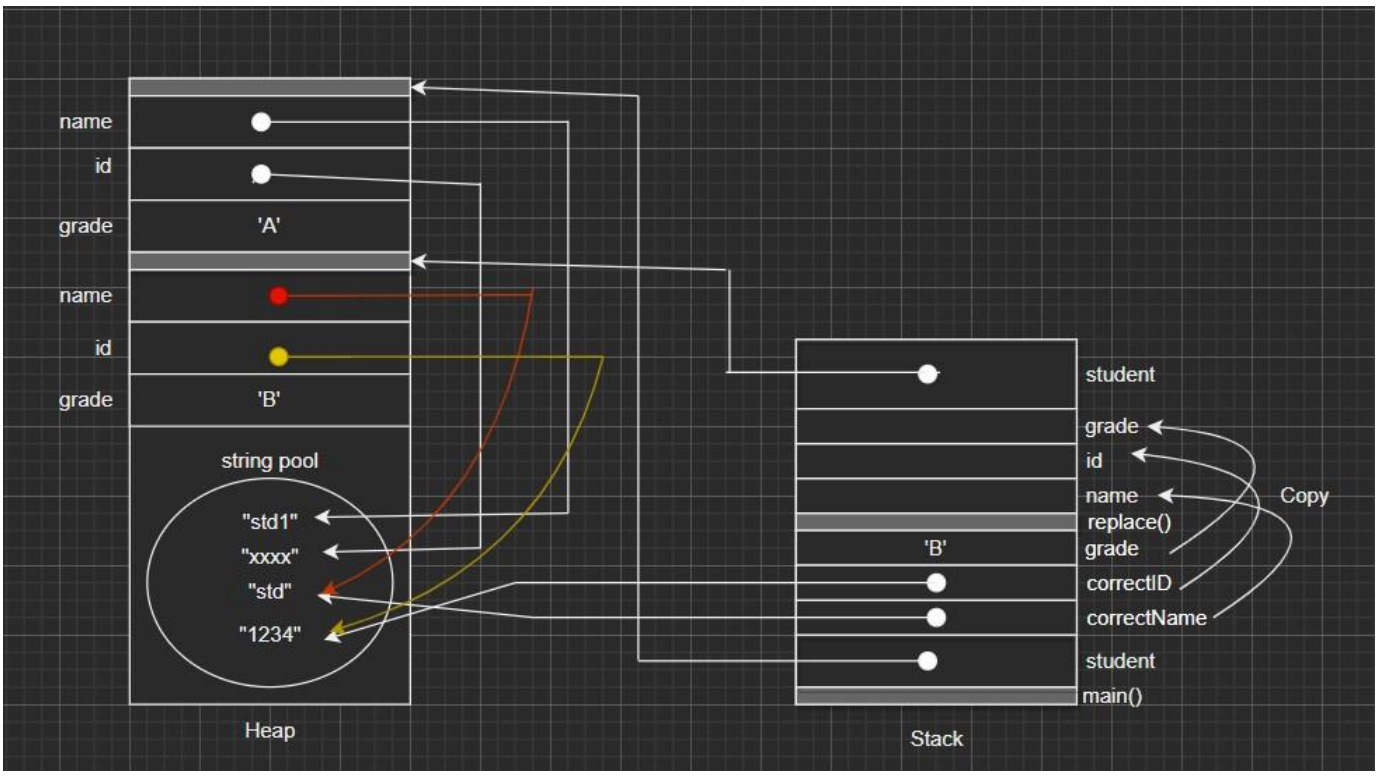
Boxing and unboxing: باکسینگ و آنباکسینگ در واقع همان تبدیل خودکار دیتاهای نوع اصلی و wapper class آن‌ها به یکدیگراست. Boxing را به تبدیل اتوماتیک دیتای نوع اصلی به wrapper class آن و unboxing را به عکس این عمل که به صورت خودکار توسط جاوا انجام می‌شود می‌گوییم.

ب)

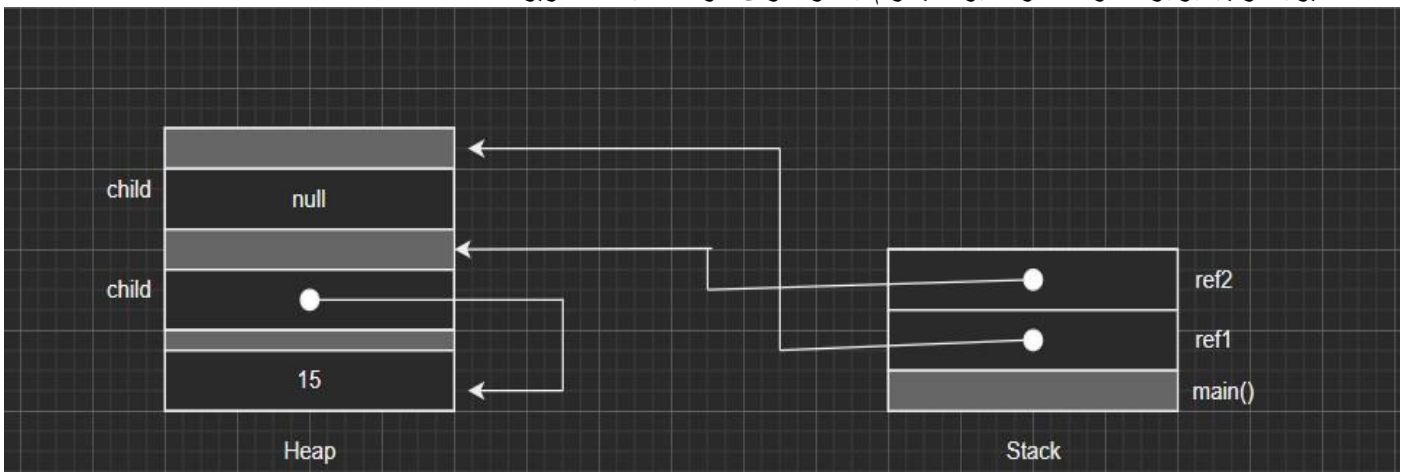
1- خیر. گاربج‌کالکتور وظیفه‌ی حذف دیتاهای بدون استفاده و بدون رفرنس را بر عهده دارد ولی مشخصا این کار نمی‌تواند تضمینی بر کم نیاوردن حافظه‌ی برنامه باشد. اگر حجم حافظه‌ی برنامه‌ی ما از مقداری که برای آن تعیین شده است بیشتر شود، حتی اگر در طول اجرای آن نیز گاربج‌کالکتور عمل کرده باشد، باز هم حافظه کم می‌آورد.
2- وقتی که ما از راه معمول و متداول ساخت شی در جاوا یعنی با استفاده از new keyword یک آبجکت جدید از کلاسی می‌سازیم، در واقع یک نمونه از آن کلاس که در فایل جداگانه‌ای است ساخته‌ایم. در ابتدای ساختن شی ابتدا حافظه‌ی لازم برای آن آبجکت در فضای heap تخصیص‌داده‌شده به برنامه گرفته می‌شود، سپس تابع constructor موجود در آن کلاس صدا زده می‌شود و اگر مقداردهی و پارامتری در آن وجود دارد انجام می‌شود. معمولا ما با استفاده از یک local variable یا یک object variable موجود در یک کلاس، رفرنسی که به آن شی که در واقع اشارەگری به اول حافظه‌ی گرفته‌شده برای آن شی در فضای heap است را نگه می‌داریم تا در آینده بتوانیم از آن شی و متدهای آن استفاده کنیم.
3- یک روش برای پیمایش بر روی یک لیست استفاده از for-rach است که در آن هر بار رفرنس یک شی موجود در آن لیست را در یک متغیر ذخیره کرده و سپس با استفاده از آن متغیر که در واقع اشارەگری به آن شی است، با آن شی کار کرده و به پیمایش ادامه می‌دهیم. یک راه دیگر استفاده از حلقه‌ی for معمولی است که از اندیس مورد نظر شروع می‌کنیم و با استفاده از index موجود در هر مرحله به اشیای آن لیست دسترسی پیدا می‌کنیم. راه سوم ما برای این کار استفاده از iterator است که با استفاده از ساخت یک iterator بر روی لیست و سپس استفاده از دو متود .next() و hasNext() روی لیست پیمایش می‌کنیم. راه چهارم ما هم می‌تواند استفاده از stream.forEach() باشد. برای مثال اگر متغیر رفرنس ما نامش list باشد می‌توان با `list.stream().forEach(System.out::println);` المان‌های لیست را چاپ کرد.
4- می‌دانیم که stack قسمتی از حافظه و یکی از چند قسمتی است که به برنامه‌هایی که با زبان‌های مختلف می‌نویسیم اختصاص داده می‌شود. به استیک به اختصار FILO نیز گفته می‌شود که اختصار عبارت Last Out ,First In می‌باشد. دلیل آن نیز از روی عبارت مشخص است. چیزهایی که به این قسمت از حافظه وارد می‌شوند بر عکس ترتیب ورودشان از آن خارج می‌شوند. برای مثال اگر دو تابع A و B داشته باشیم که در تابع A تابع B را صدا بزنیم، ابتدا frame تابع A در استک گرفته می‌شود، سپس تابع B وارد می‌شود و پس از اتمام کار تابع B، ابتدای فضای تابع B و سپس پس از اتمام تابع A فضای تابع A آزاد می‌شود. همچنین این قسمت از حافظه محل نگه‌داری متغیرهای local نیز می‌باشد. پس می‌توان حدس زد که یکی از مواقعی که احتمال اتفاق افتادن stackOverFlow بالاست مواقعی است که در تعداد زیادی تابع هر بار تابع یا توابعی را به صورت متوالی صدا می‌زنیم. این کار باعث می‌شود که هر بارمقدار بیشتری از حافظه برای توابع جدید گرفته شود و ممکن است منجر به این پدیده شود. مثال خوب برای آن را می‌توان استفاده از توابع بازگشتی دانست به خصوص زمانی که شرط اتمام فراخوانی و پایان تابع را به درستی تنظیم نکرده باشیم و باعث شویم بی‌نهایت بار یک تابع خودش را صدا کند. بهتر است همیشه به call tree که در واقع درخت فراخوانی توابع مختلف در برنامه است توجه داشته باشیم که در مقاطعی از برنامه بیش از اندازه شلوغ و چندشاخه نشود که منجر به این مشکل شود.
5- یکی از تفاوت‌های اصلی بین این دو کلاس این است که در کلاسHashMap ما در هر بار اد کردن یک key و یک value را اضافه می‌کنیم و در واقع با این کار هر کلید را به یک مقدار مپ می‌کنیم. نکته‌ی مهم این است که ممکن است که در یک آبجکت از این کلاس value های تکراری وجود داشته باشد ولی keyهای یکسان نمی‌تواند وجود داشته باشد. ولی در HashSet ما دسته‌ای از اشیا را در یک آبجکت از این کلاس نگه می‌داریم و با هر بار اضافه کردن نیز یک آبجکت اضافه می‌کنیم و مهم‌ترین نکته در آن این است که نمی‌توانیم یک valueی یکسان در آن داشته باشیم. همچنان باید برای ساختن یک HashSet شامل یک شی، توابع hashCode و equals را نیز override کنیم تا مانع از اضافه شدن اشیای تکراری به آن شویم. از لحاظ سرعت نیز می‌توان گفت سرعت کار با HashSet بیشتراز سرعت HashMap است زیرا از تکنیک hashing استفاده می‌کند.

ج)

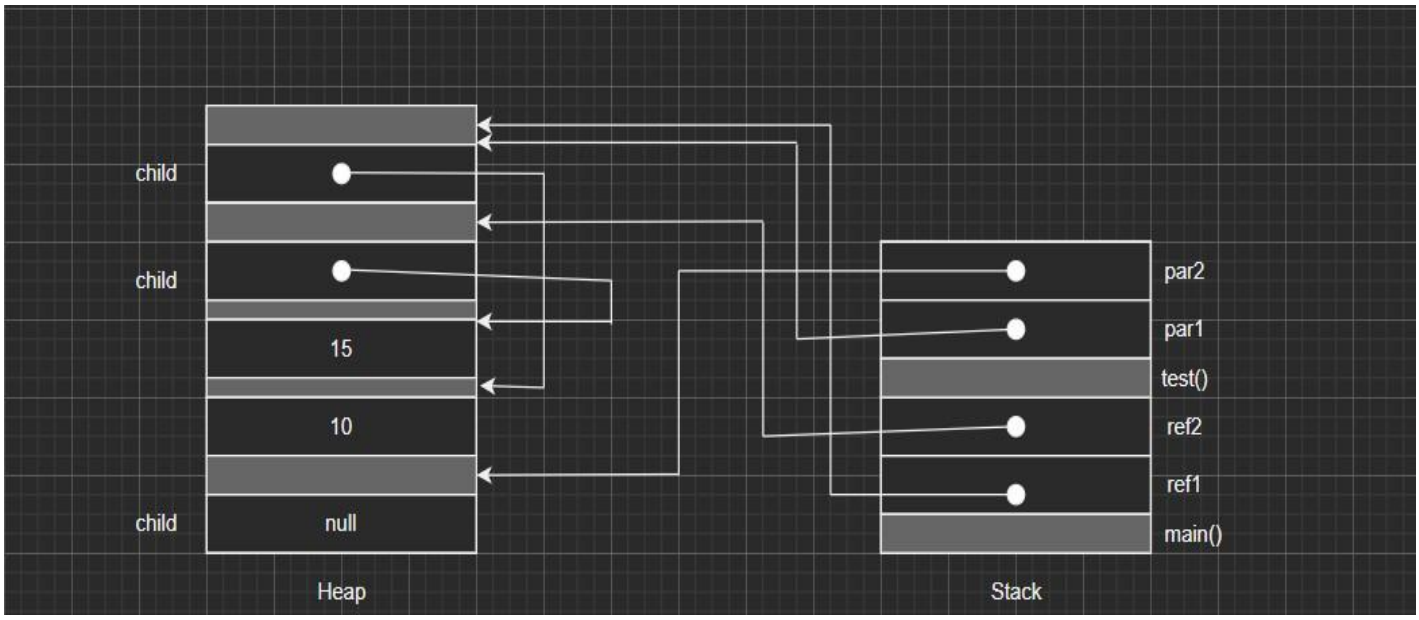
1- خیر، مقادیر تغییر نمی‌کند زیرا ما در تابع replace صرفا یک شی جدید با مقادیر جدید ساخته‌ایم و آن را به رفرنس local آن داده‌ایم. پس پس از خروج از replace تفاوتی در مقادیر شی اولیه ایجاد نمی‌شود. دیاگرام heap-stack تا انتهای اجرای متود replace به شکل زیر است:



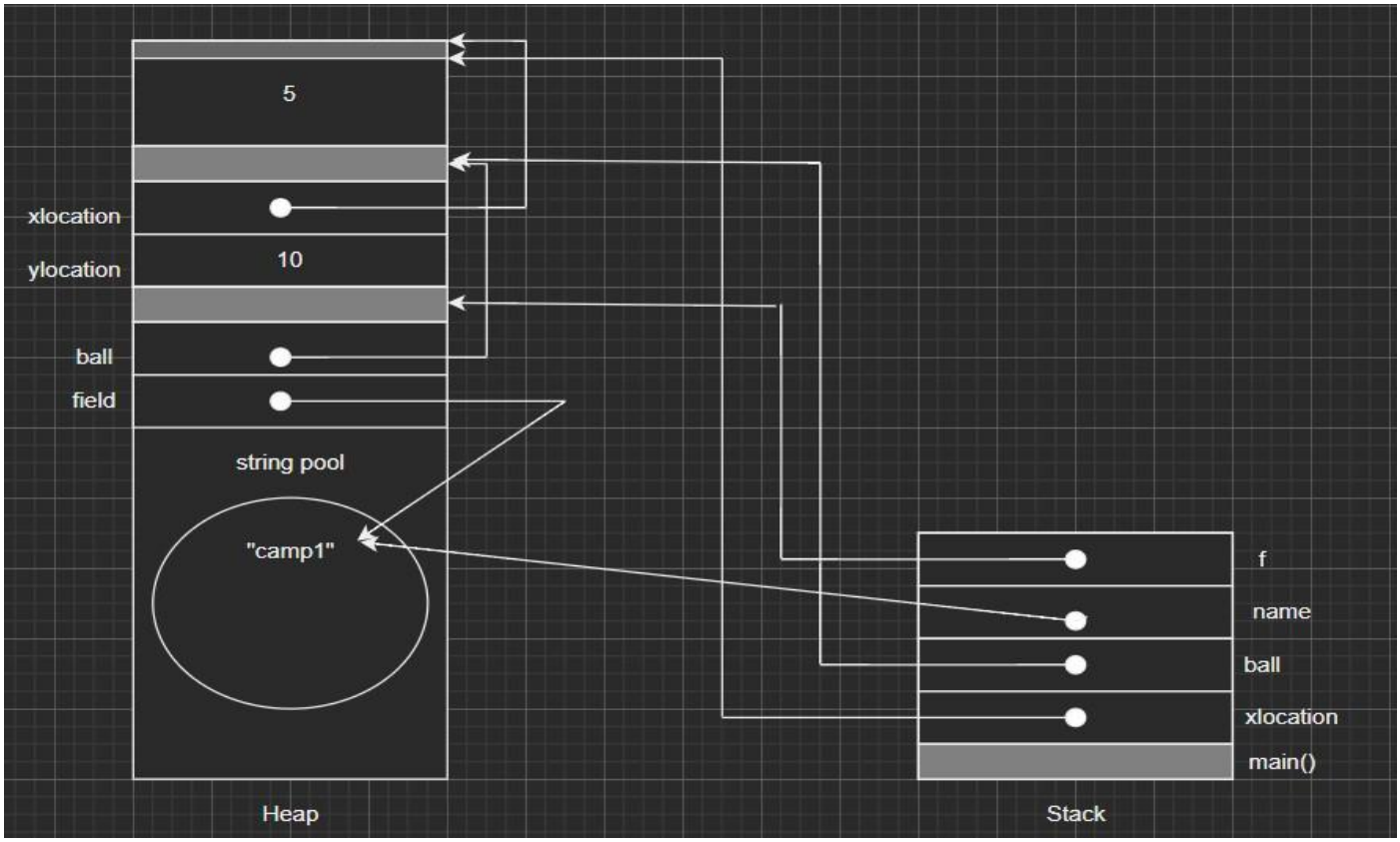
2- مقادیر به ترتیب برابر 10 و 15 خواهد بود. دیاگرام قبل از اجرای متود test به شکل زیر:



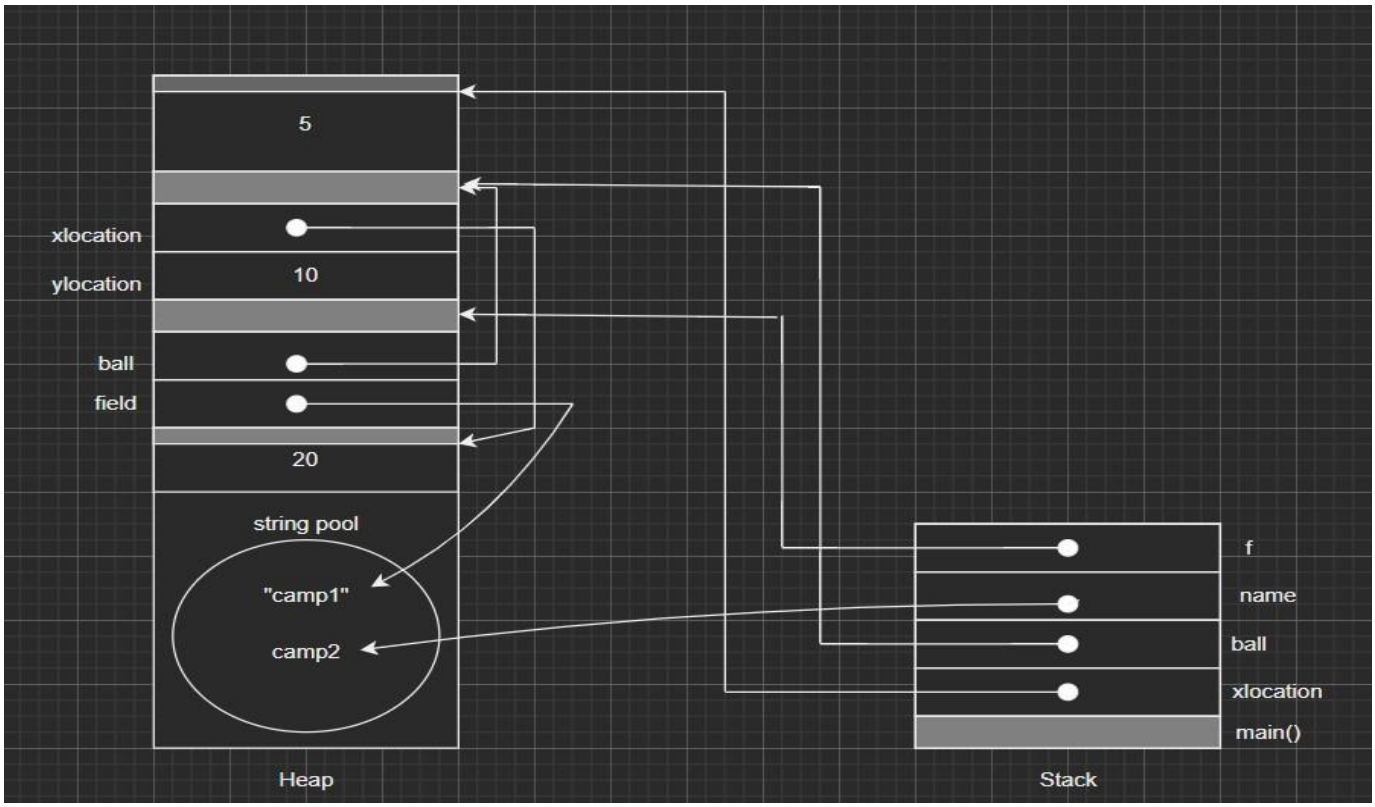
و در انتهای اجرای متود test به شکل زیراست:



3- مقادیر خواسته شده در سوال به ترتیب 20, camp1, camp2 هستند. دیاگرام تا قبل از اجرای متد set به شکل زیر:



و پس از اجرای متود ست و اجرای دستور "name = "camp2 به صورت زیر است:



د)

مقدار حافظه‌ی آزاد قبل از فراخوانی gc برابر با 4232303632 و پس از آن برابر 4233632688 بود و در نتیجه پس از فراخوانی gc درواقع 1329056 بایت فضا آزاد شده است.

در حین اجرای برنامه هر چند مدت یکبار garbage collector به صورت اتوماتیک صدا می‌شود. نحوه‌ی عملکرد آن به این صورت است که هر بار با فراخوانی آن، قسمت‌هایی از حافظه که اشغال شده‌اند ولی ما دسترسی به آن‌ها نداریم(رفرنسی برای استفاده از آن‌ها نداریم) را آزاد می‌کند و این کار منجر به افزایش حافظه‌ی free برنامه‌ی ما و بهبود عملکرد و سرعت آن می‌شود.