فرهاد امان 9931006

1)

الف)

گاهی نیاز پیدا می کنیم که نوع های اولیه را به صورت یک شی داشته باشیم در این مواقع از wrapper class ها استفاده می کنیم به عنوان مثال در collection ها ما نیاز داریم که از آن ها استفاده کنیم فرآیند تبدیل primitive type ها به اشیاء به صورت خودکار صورت می گیرد که به آن اصطلاحا autoboxing و به روند برعکس آن یعنی تبدیل شی به primitive type , unboxing می گویند.

هنگامی که اشیائی در حافظه heap تعریف میکنیم باید reference هایی تعریف کنیم که آدرس آن شی را نگه دارند هنگامی که هیچ reference به یک شی نداشته باشیم به آن شی اصطلاحا garbage گفته می شود جاوا به صورت خودکار این اشیاء را پیدا میکند و اقدام به حذف کردن آن ها از حافظه heap میکند تا کارایی حافظه افزایش یابد.

ب)

1. خیر ابتدا باید این موضوع را در نظر بگیریم که garbage collector تنها اشیائی را حذف میکند که هیچ reference به آن ها وجود ندارد پس میتوانیم تعداد زیادی شی reference دار بسازیم و باعث کمبود حافظه شود موضوع دوم این است که garbage collector با تاخیر عمل میکند و بالافاصله قادر به پاک کردن garbage ها نیست
2. هنگامی که ما یک شی جدید را new میکنیم در حافظه heap یک فضا در نظر گرفته می شود که تمام متغیر ها و حافظه مورد نیاز برای آن شی را در بر میگیرد همچنین یک over head برای نگه داری حافظه مورد نیاز خود جاوا نیز در نظر گرفته می شود در نهایت ما با ایجاد یک reference یک فضا در حافظه stack ایجاد میکنیم که این فضا مسئول نگه داری آدرس ابتدای شی مورد نظر ما در حافظه heap می باشد.
3. روش اول استفاده فور و یک متغیر از نوع int که از 0 تا کمتر از list.size() حرکت میکند . روش دوم از استفاده از for each می باشد که به صورت خودکار تمام اعضای لیست را طی میکند . روش بعدی استفاده کلاس Iterator می باشد که با استفاده از یک حلقه while و متد های it.hasNext() و it.next() می توانیم لیست را طی کنیم روش بعدی از استفاده از حلقه for each داخلی خود لیست است که به صورت list.forEach() استفاده می شود.

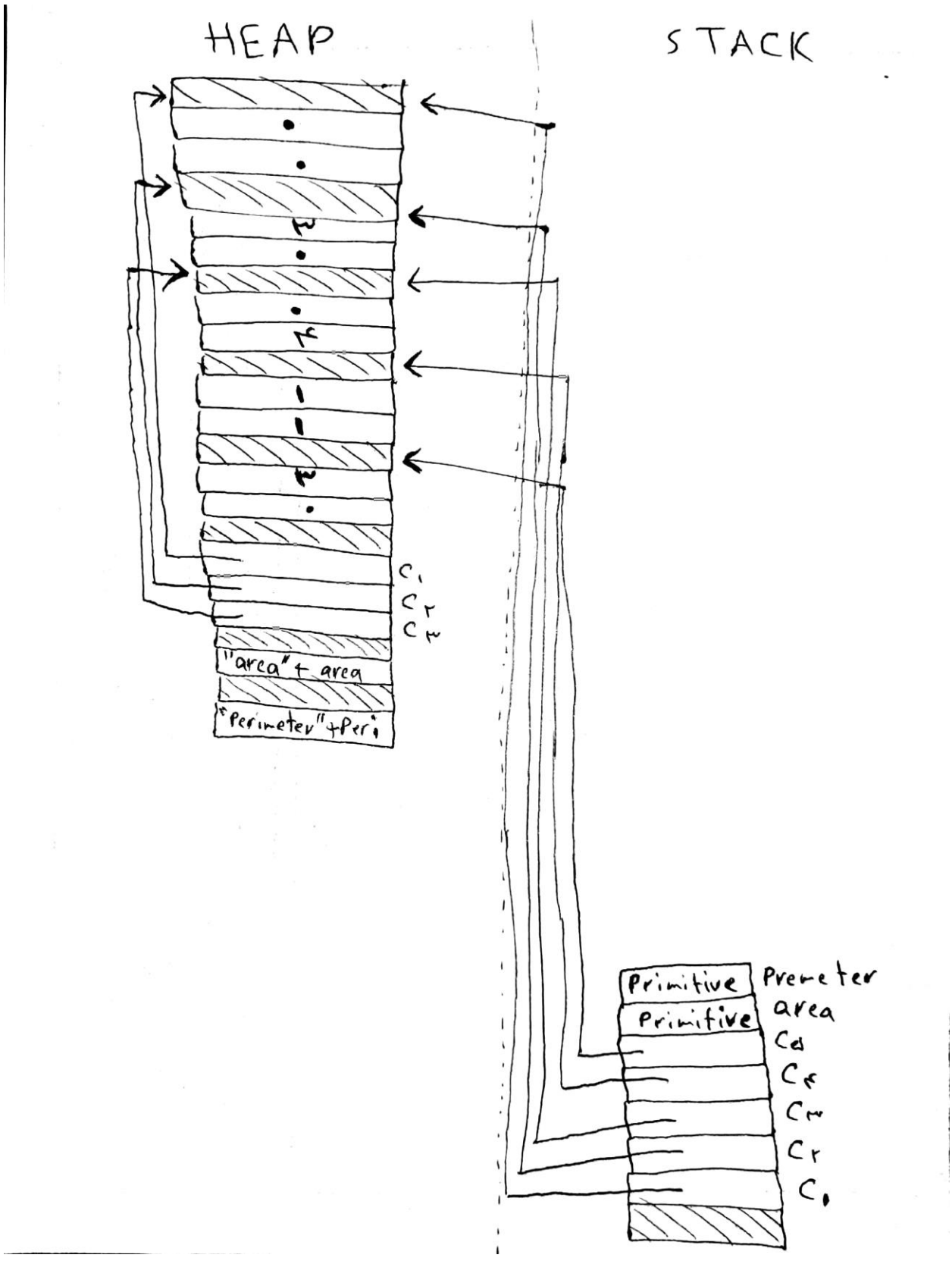
4. همانطور که میدانیم جاوا از دو حافظه heap و stack استفاده میکند . این دو حافظه از دو طرف مموری و در جهت عکس همدیگر شروع به مقدار دهی میکنند و به هم نزدیک تر میشوند . هنگامی که به خاطر رشد شدید حافظه stack مقادیر این حافظه با مقادیر حافظه heap به هم برخورد کنند stack overflow اتفاق می افتد . این مشکل معمولا هنگام استفاده از توابع بازگشتی اتفاق می افتد به تعداد زیاد یا بدون محدودیت خود را صدا میزنند. در این صورت stack frame های بسیار زیاد برای این متود ها ایجاد شده و باعث این مشل می شوند.
5. در HashMap ما داده ساختار map را با استفاده از یک hash table پیاده سازی کرده ایم درحالیکه در HashSet ما set را با استفاده از آن پیاده سازی کرده ایم . در HashSet امکان نگه داری عناصر تکراری وجود ندارد در حالیکه در HashMap کلید ها منحصر به فرد و مقادیر میتوانند یکسان باشند درست مانند یک تابع ریاضی . در واقع ما در HashSetتنها تعدادی شی را نگه داری میکنیم که تمام آن ها متفاوت اند و میتوانند ترتیب خاصی هم نداشته باشند و indexed نیستند اما در HashMap ما جفت هایی از از اشیاء را نگه داری میکنیم که با استفاده از از عضو اول هر جفت میتوانیم به جفت دوم آن دسترسی داشته باشیم .

(2

1. خیر پنهان سازی اطلاعات باعث کم شدن وابستگی اشیاء به یکدیگر می شود و باعث افزایش level of dependence می شود.
2. خیر در package access تنها کلاس های داخل همان پکیج به آن دسترسی خواهد داشت .
3. خیر چنین امکانی وجود ندارد چون در این صورت تعریف آن کلا فایده ای ندارد.
4. خیر در map امکان ایجاد کلید های یکسان وجود ندارد ولی امکان داشتن value های یکسان وجود دارد.
5. خیر کاملا برعکس جاوا همیشه pass by value می باشد. حتی در هنگام کار با اشیاء ما در واقع آدرس شی را پاس می دهیم .
6. بله در این صورت امکان ایجاد یک شی از آن کلاس خارج از خود کلاس وجود نخواهد داشت .

(4

الف)


HEAP
STACK
"area" + area
"Perimeter" + Peri
Primitive
Primitive
Preme ter
area

بله String هایی که هنگام پاس داده شدن به توابع println ایجاد شده اند garbage هستند و بعد از پایان متود println هیچ reference ای به آنها وجود ندارد.

ب) جواب دوم و چهارم حتما true خواهد چرا که از متد equals استفاده شده و مقدار string ها با هم مقایسه می شود جواب چهارم حتما false می باشد چرا که str3 وstr4 با اینکه دارای مقدار یکسان می باشند اما عملگر== تنها برابری آدرس های آن ها را بررسی میکند. جواب اول میتواند true باشد که جاوا دارای روشی برای بهینه سازی میباشد. که اگر چند String با مقدار یکسان با روش = قرار دادن ساخته شدند به جای ایجاد دوباره هر کدام و گرفتن فضای متفاوت در heap تنها یکبار فضا را میگرید و آدرس آن را به reference ها میدهد.