الف)

Wrapper class : در زبان های مدرن تر برنامه نویسی نیاز به داشتن یک سری تابع کمکی برای هر کدام از primitive type ها یا حتی non-primitive ها نیاز داریم. این توابع باید توسط خود زبان نوشته شده و maintain شوند. با توجه به خاصیت شی گرایی محضی که در زبان java برقرار است راه حلی که برای primitive type ها انتخاب شد این بود که یک class برای هر کدام از آنها داشته باشیم که ما اون primitive را به آن می دهیم و بعد آن class احتمالا در یکی از filed هایش ان را ذخیره میکند. این ویژگی این اجازه را برنامه نویس های هسته ی زبان جاوا می دهد که تابع های کمکی که نیاز به پیاده سازی آن دارند در واقع به صورت یک method از آن wrapper class ؛ که در واقع دور ان primitive type را گرفته است و به نوعی ان تبدیل astroides primitive data type کرده است ؛ تعریف کنند.

AutoBoxing and Unboxing: با توجه به مطالبی که در بالا اشاره شد در واقع هر کدام از Wrapper Class ها یک class هستند که خود برنامه نویس های هسته ی جاوا ان را تعریف کرده اند پس در واقع هر وقت نیاز به استفاده از آن تابع کمکی داریم باید یک نمونه ی جدید از ان wrapper class ایجاد کنیم اما این کار به نوعی کمی cumbersome به نظر میرسد پس در سطح java compiler این ویژگی ایجاد شد که در ان موقع نیاز داریم خود compiler ان class که نیاز دارد را دور primitive data type درست کند و آن موقع هم که آن را نیاز نداریم آن را بیرون بیاورد. به این عمل autoboxing and unboxing میگوییم.

Garbage collection : با توجه به مطالبی که در مبانی نرم کامپیوتر بیان شد هر process که در سیستم ما انجام می شود نیاز به مقدار مشخصی از منابع سخت افزاری کامپیوتر دارد و بدیهی است که نرم افزار هایی که با زبان java نوشته شده اند نیز از این قاعده مستثنا نیستند. حال فرقی که زبان java با بعضی از زبان های قبل از خودش ایجاد کرده است این است که ابزاری به نام garbage collector در jvm خود دارد که مرتب اشیایی که در heap قرار دارند را در نظر دارد و تعداد link هایی که به آنها از قسمت static ها یا stack هست می شمارد و هر گاه این شمارش به عدد صفر رسید به این معنا که هیچ متغیری نیست که نیاز به ان شیا در heap داشته باشد garbage collector وارد عمل میشود و آن قسمت از heap را خالی می کند که استفاده از منابعی که نرم افزار دارد را به حداقل برساند.

ب)

ج۱: خیر ؛ همانطور که در موارد قبلی هم اشاره شد garbage collector یک ابزار است و بیشتر کار خود را در heap انجام میدهد و حتی زمانی وارد عمل می شود که هیچ pointer یی به ان شی ذخیره شده موجود نباشد. حال اگر این سناریو را در نظر بگیریم که ما تعداد خیلی خیلی زیادی شیا داریم که نیاز به همه ی آنها داریم و برای همه ی آنها pointer موجود است دیگر garbage collector اصلا وارد عمل نمی شود و اگر تعداد این اشیا بسیار بسیار زیاد باشد امکان دارد سیستم حافظه کم بیاورد و سیستم عامل kill process انجام دهد.(این نکته هم حائز اهمیت است که این مثال تعداد زیادی شیا تنها یک حالت از کل حالت هایی است که امکان دارد پیش بیاید و garbage collector نتواند هیچ کاری کند و سیستم حافظه کم بیاورد و بیشتر هدف از مطرح کردن ان اوردن یک مثال برای بهتر روشن کردن موضوع بود)

ج۲ : همانطور که در موارد قبلی نیز توضیح داده شد در واقع هر process نیازمند مقداری از ram است . حال زبان java این مقدار ram را به ۳ تیکه تقسیم میکند که قسمت اول static memory هست که متغیرهای static یا method های استاتیک را در آن قرار می دهد و یک قسمت stack هست که در مورد آن به صورت مفصل در مبانی کامپیوتر توضیح داده شد و قسمت بعدی heap است که در واقع jvm تمام اشیا را در آنها میسازد و ذخیره می کند. حال هنگامی که ما یک Object می سازیم در واقع jvm قسمتی از heap را به آن object اختصاص میدهد و refrence ان را در ان متغیری که ما در stack برای ان ساخته ایم ذخیره می کند و این امکان را فراهم میسازد که از طریق ان به behaviour های ان شی دسترسی پیدا کرد

ج۳:

1. استفاده از loop معمولی که با استفاده از یک index counter در list جلو میرویم ولی در این مورد نمی توان هیچ تغییری در خود لیست ایجاد کرد و فقط می توان item ها را از list بیرون کشید و با انها کاری انجام داد.
2. استفاده از enhanced for loop که گاها foreach هم اسم گذاری می شود که در این مورد به جای اینکه ما به صورت دستی یک index counter داشته باشیم jvm یک متغیر برای ما می سازد و به ترتیب item ها را از list بیرون میکشد اما باز هم محدودیت های بالا را داریم.
3. استفاده از iterator که یک private property از خود class هست و می توان با استفاده از مشکلاتی را که در دو مورد قبلی داریم را حل کرد.
4. Foreach method : در این حالت که یه مدت از خود list به اسم foreach که به این یک callback میدهیم و ان callback بر تمام اعضای ان list اجرا میشود و خروجی ان در یک list جدید قرار گرفته می شود و return میشود.

ج۴: پدیده ی stack overflow در واقع یعنی زمانی که یک نرم افزار اقدام به گرفتن حافظه ی بیشتر از حدی که سیستم عامل به ان اجازه داده است یا به صورت کلی بیشتر از ان میزانی که به صورت

سخت افزاری در دست نرم افزار است ؛ می کند. به صورت کلی می توان گفت که stack overflow به راحتی قابل detect کردن و debug کردن نیست و چیز های زیادی امکان دارد این مشکل را به وجود آورده باشند. یک راه برای پیشگیری میتواند استفاده از primitive type های بهتر و کم حجم تر در صورت امکان است(منظور این است که به جای اینکه از long استفاده کنیم اگر عدد کوچک است از int یا حتی short استفاده کنیم) ؛ یک راه دیگر میتواند استفاده از سخت افزار قوی تر باشد و راه متعدد دیگری که همه ی آنها باید بررسی شوند .

ج۵: در واقع HashSet و HashMap یک نوع data structure بسیار مهم در دنیای نرم افزار هستند که در یکی از آنها بر اساس مفهوم set کار میکنیم و در دیگری بر اساس Map. در HashSet به صورت کلی یک آرایه ی بسیار بزرگ داریم که هر وقت میخواهیم یک item را از آن بیرون بیاوریم یا در آن ذخیره کنیم یک تابع hash صدا میشود و در نهایت یک index خروجی داده میشود که آن داده در آن در صورتی که تکراری نباشد ذخیره می شود پس می توان گفت در واقع hashSet ها بر اساس index در آن ارایه هستند هر چند در واقع ها هیچ دسترسی به آن آرایه ی داخلی و index های آن نداریم. از طرفی دیگر HashMap به این صورت نیست و بر اساس index نیست و بر اساس key است که به آن می دهیم به این معنا که هر کدام از iterm هایی که به صورت key به آن می دهیم را به یک value ربط میدهد و این کار را به کمک همان تابع hash و آرایه ها می کند اما روش آن کمی با hashSet متفاوت است.