viettel
software services

CHƯƠNG TRÌNH ĐÀO TẠO JAVA NÂNG CAO

Bộ nhớ trong Java

Đơn vị tổ chức: Phòng đào tạo và phát triển nguồn nhân lực

viettel
software services

NỘI DUNG HỌC PHẦN

BÀI 1

Bộ nhớ trong Java

BÀI 2

Cấp phát và thu hồi bộ nhớ trong Java

BÀI 3

Tham chiếu và tham trị

viettel
software services

BÀI 1

Bộ nhớ trong Java

viettel
software services

MỤC LỤC

01

Stack và Heap

viettel
software services

Bộ nhớ trong Java

- Trong lập trình cũng như trong Java bộ nhớ máy tính được chia hai loại là Stack và Heap
- Khi chạy chương trình Java, JVM sẽ yêu cầu hệ điều hành cấp cho một không gian bộ nhớ trong RAM để dùng cho việc chạy chương trình. JVM sẽ chia bộ nhớ được cấp phát này thành 2 phần: Heap và Stack cho việc quản lý.

viettel
software services

Heap Memory

- Java Heap Memory là bộ nhớ được sử dụng ở runtime để lưu các Objects. Bất cứ khi nào ở đâu trong chương trình của bạn khi bạn tạo Object thì nó sẽ được lưu trong Heap (thực thi toán tử new).
- Các objects trong Heap đều được truy cập bởi tất cả các các nơi trong ứng dụng, bởi các threads khác nhau.
- Thời gian sống của object phụ thuộc vào Garbage Collection của java.
- Garbage Collection sẽ chạy trên bộ nhớ Heap để xoá các Object không được sử dụng nữa, nghĩa là object không được referece trong chương trình.
- Dung lượng sử dụng của Heap sẽ tăng giảm phụ thuộc vào Objects sử dụng.
- Dung lượng Heap thường lớn hơn Stack.

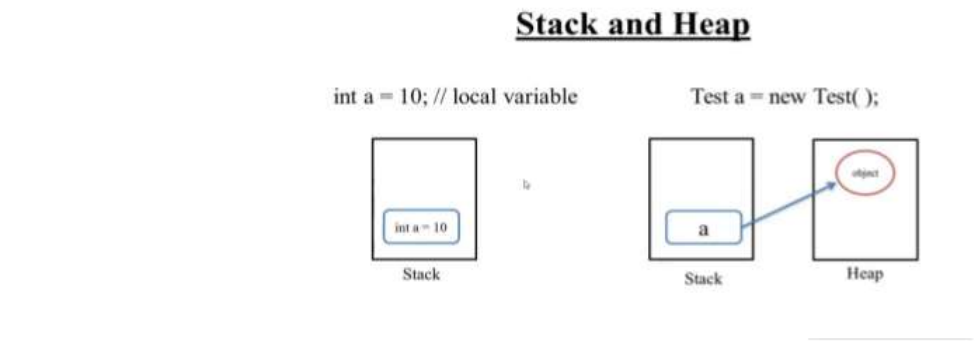
viettel
software services

Stack Memory

- Bộ nhớ để lưu các biến local trong hàm.
- Các biến local bao gồm loại nguyên thuỷ (primitive) và loại tham chiếu tới đối tượng trong heap (reference) khai báo trong hàm, hoặc đối số được truyền vào hàm, thường có thời gian sống ngắn.
- Bộ nhớ stack thường nhỏ.
- Cơ chế hoạt động thức của Stack là những phương thức , biến chạy sau thì sẽ bị giải phóng đầu tiên
- Khi hàm được gọi thì một vùng nhớ được tạo ra trong stack và lưu các biến trong hàm đó. Khi hàm thực hiện xong, khối bộ nhớ cho hàm sẽ bị xóa, và giải phóng bộ nhớ trong stack.

viettel
software services

Stack và Heap



viettel
software services

02

Sự khác nhau giữa Stack và Heap

viettel
software services

Heap	Stack
Java Heap Memory là bộ nhớ được sử dụng ở runtime để lưu các Objects. Bất cứ khi nào ở đâu trong chương trình của bạn khi bạn tạo Object thì nó sẽ được lưu trong Heap (thực thi toán tử new).	Stack Memory là bộ nhớ để lưu các biến local trong hàm và lời gọi hàm ở runtime trong một Thread java. Các biến local bao gồm: loại nguyên thuỷ (primitive), loại tham chiếu tới đối tượng trong heap (reference), khai báo trong hàm, hoặc đối số được truyền vào hàm.
Thời gian sống của bộ nhớ Heap dài hơn so với Stack. Thời gian sống của object phụ thuộc vào Garbage Collection của java. Garbage Collection sẽ chạy trên bộ nhớ Heap để xoá các Object không được sử dụng nữa, nghĩa là object không được referece trong chương trình.	Thường có thời gian sống ngắn.

02 – Sự khác nhau giữa Stack và Heap

viettel
software services

Heap	Stack
Các objects trong Heap đều được truy cập bởi tất cả các các nơi trong ứng dụng, bởi các threads khác nhau.	Stack chỉ được sử dụng cho một Thread duy nhất. Thread ngoài không thể truy cập vào được.
Cơ chế quản lý của Heap thì phức tạp hơn. Heap được phân làm 2 loại Young-Generation, Old-Generation. Đọc thêm về Garbage Collection để hiểu rõ hơn.	Cơ chế hoạt động là LIFO (Last-In-First-Out), chạy sau chết trước.
Dung lượng Heap thường lớn hơn Stack.	Bộ nhớ stack thường nhỏ.
Sử dụng -Xms và -Xmx để định nghĩa dung lượng bắt đầu và dung lượng tối đa của bộ nhớ heap.	Dùng -Xss để định nghĩa dung lượng bộ nhớ stack.

viettel
software services

Heap	Stack
Khi Heap bị đầy chương trình hiện lỗi java.lang.OutOfMemoryError: Java Heap Space	Khi stack bị đầy bộ nhớ, chương trình phát sinh lỗi: java.lang.StackOverFlowError
Truy cập vùng nhớ Heap chậm hơn Stack.	Truy cập Stack nhanh hơn Heap
Dung lượng sử dụng của Heap sẽ tăng giảm phụ thuộc vào Objects sử dụng.	Bất cứ khi nào gọi 1 hàm, một khối bộ nhớ mới sẽ được tạo trong Stack cho hàm đó để lưu các biến local. Khi hàm thực hiện xong, khối bộ nhớ cho hàm sẽ bị xoá, và giải phóng bộ nhớ trong stack.

03

Ví dụ

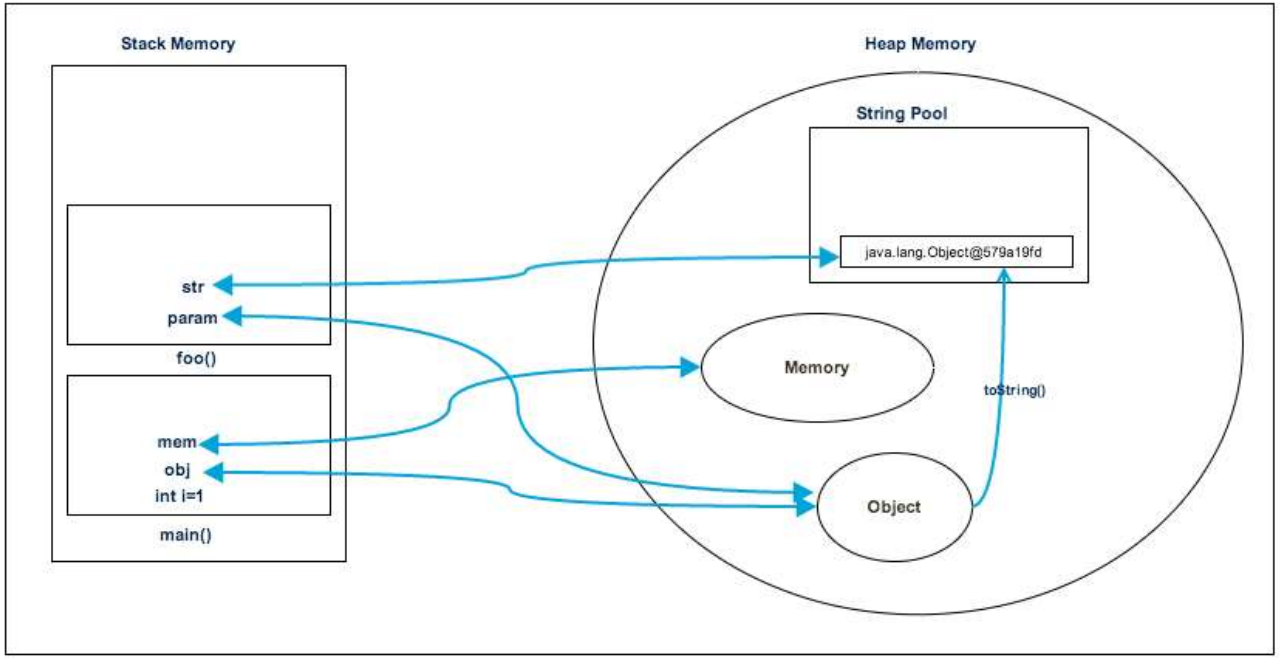
viettel
software services

Ví dụ

```
1  public class Memory {
2
3      public static void main(String[] args) { // Line 1
4          int i=1; // Line 2
5          Object obj = new Object(); // Line 3
6          Memory mem = new Memory(); // Line 4
7          mem.foo(obj); // Line 5
8      } // Line 9
9
10     private void foo(Object param) { // Line 6
11         String str = param.toString(); //// Line 7
12         System.out.println(str);
13     } // Line 8
14
15 }
```

viettel
software services

Ví dụ



viettel
software services

BÀI 2

Cấp phát và thu hồi bộ nhớ

viettel
software services

MỤC LỤC

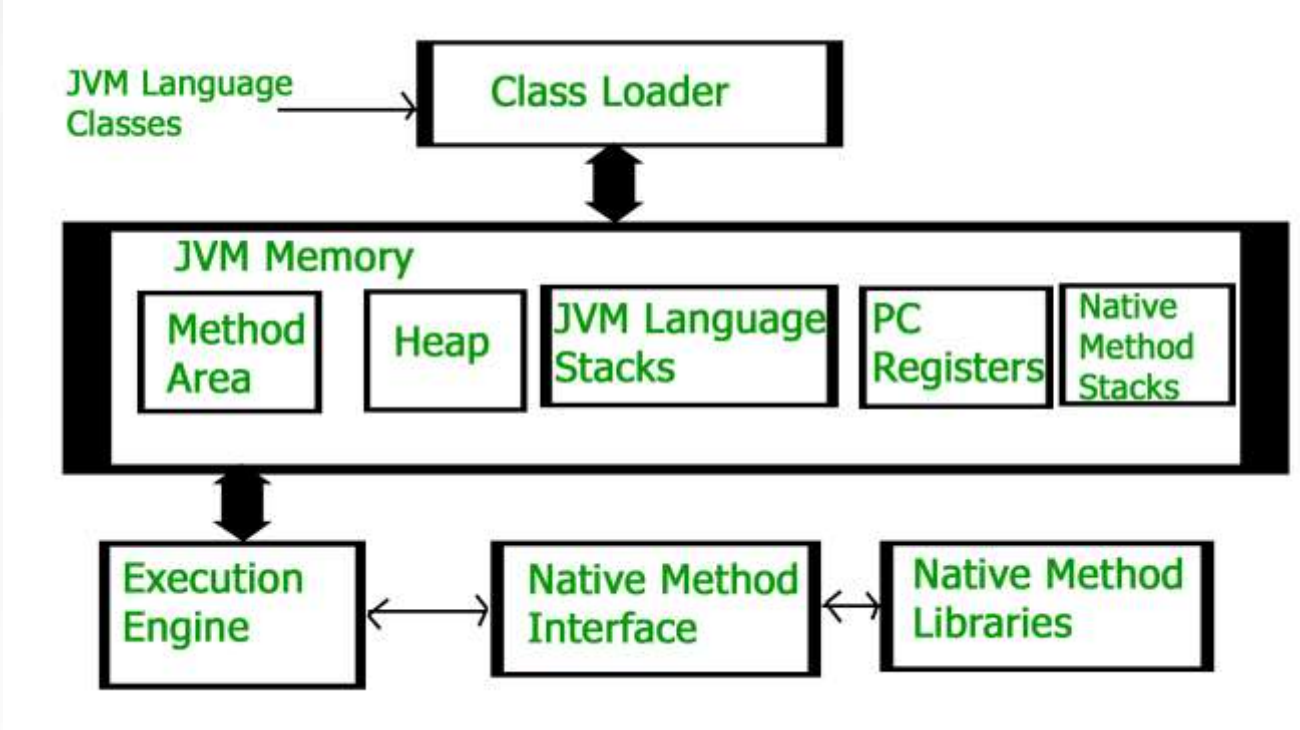
viettel
software services

01

Kiến trúc bộ nhớ trong JVM

viettel
software services

Kiến trúc JVM



viettel
software services

Kiến trúc bộ nhớ trong JVM

- **Method area:** Giúp lưu trữ dữ liệu cho mỗi lớp (class). Mỗi lớp thì có dữ liệu gì? Đó có thể là tên lớp, lớp đó có access modifier gì, các thuộc tính và phương thức của lớp, các đoạn code của phương thức,…Lưu ý, vùng nhớ này lưu thông tin về lớp (class), chứ không phải đối tượng (object) của lớp (class).
- **Program Counter Register:** Lưu trữ địa chỉ thực thi hiện tại của một luồng. Mỗi luồng sẽ có PC Register riêng biệt
- **Native Method Stack:** Mỗi luồng sẽ được cung cấp một Native Stack riêng biệt. Nó sẽ lưu trữ thông tin của phương thức gốc (phương thức được viết bằng ngôn ngữ khác)
- **Stack và Heap**

02

Heap và Nursery

viettel
software services

Giải phóng bộ nhớ trong Heap

- Các đối tượng java nằm bên trong một vùng nhớ gọi là heap. Heap được tạo ra khi JVM start up và có thể tăng hoặc giảm kích thước khi các ứng dụng chạy. Khi heap trở nên đầy, quá trình thu gom rác (GC) sẽ chạy, lúc này các đối tượng không còn được sử dụng sẽ bị xóa đi, tạo ra khoảng trống cho các đối tượng mới trên heap.

viettel
software services

Heap và Nursery

- Heap thường được chia làm 2 vùng gọi là Nursey (hay young space) và vùng old space.
- Nursery là một phần của heap để dành cho việc cấp phát cho các đối tượng mới. Khi nursery bắt đầu đầy, rác sẽ được thu gom bằng một process thu gom rác đặc biệt gọi là young collection, nơi mà các đối tượng sống đủ lâu trong nursery được di chuyển lên vùng old space, do đó giải phóng nursery để cấp phát các đối tượng khác.
- Khi old space trở nên đầy, rác sẽ được thu gom bởi process khác được gọi là old collection.
- Một process young collection được thiết kế để nhanh chóng tìm các đối tượng mới được cấp phát mà vẫn tồn tại (alive) và di chuyển chúng khỏi nursery. Thông thường, young collection process sẽ giải phóng một số lượng bộ nhớ nhất định nhanh hơn old collection process

03

Garbage collection

viettel
software services

Garbage Collection

- Máy ảo JVM trong Java được sử dụng phổ biến là Java HotSpot. Java HotSpot có nhiều chương trình Garbage Collection (GC) chạy nền trong nó. Có 4 bộ GC khác nhau trong Java HotSpot: **Serial, Parallel, Concurrent Mark Sweep (CMS), Garbage First (G1)**
- Garbage collection là chương trình chạy nền, nó theo dõi toàn bộ các Object trong bộ nhớ (Heap) và tìm ra những Object nào không được dùng nữa (là các đối tượng không được tham chiếu đến nữa).

viettel
software services

Garbage Collection

- Có 3 trường hợp đối tượng không được tham chiếu nữa:

Gán null

```
Employee e=new Employee();
e=null;
```

Gán đến một tham chiếu khác

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;
```

Đối tượng anonymous

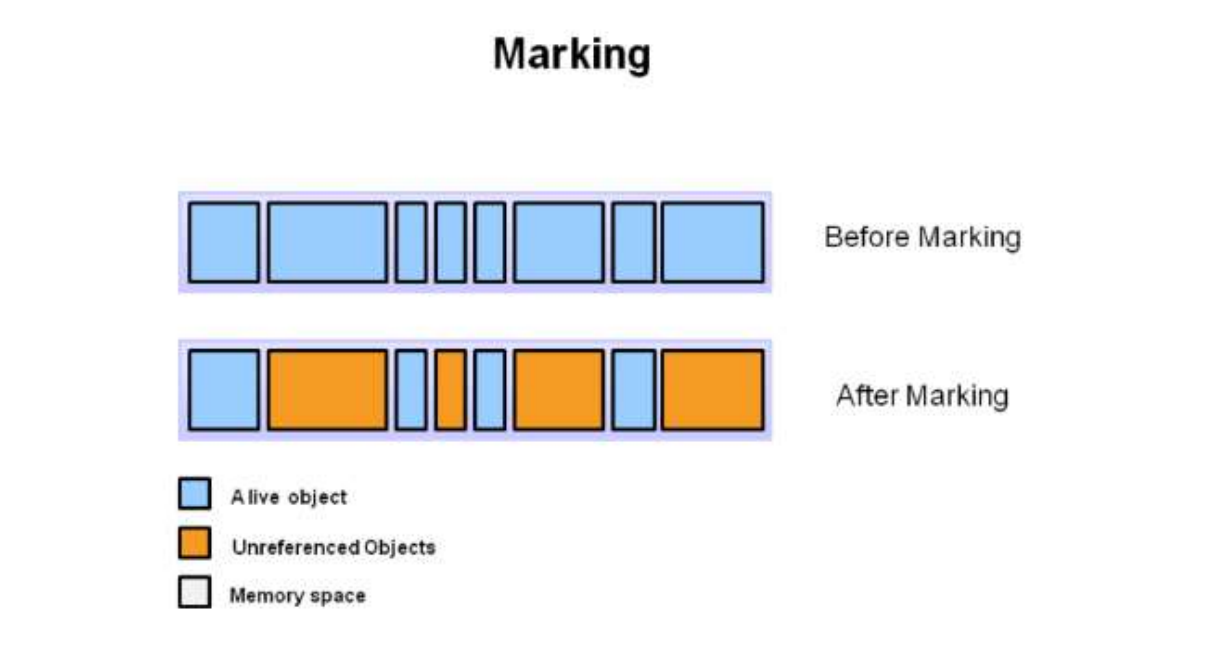
```
new Employee();
```

Ngoài ra, một đối tượng cũng có thể trở thành "rác" nếu không còn bất kỳ tham chiếu nào tới nó từ bất kỳ đối tượng hoặc biến nào khác trong chương trình

viettel
software services

Mô hình hoạt động

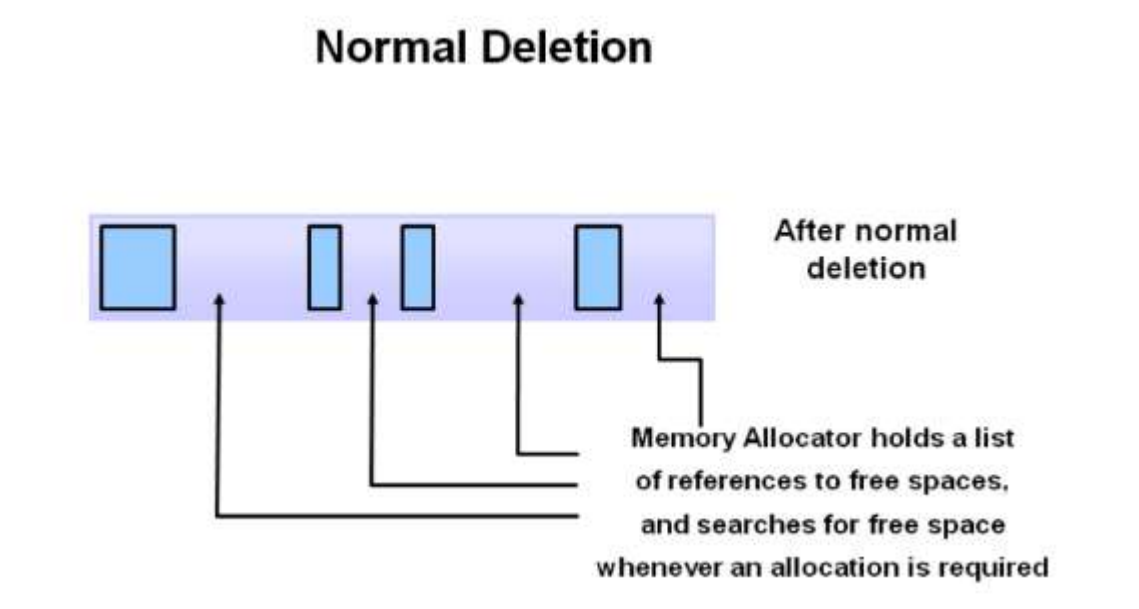
- **Bước 1 Marking:** Là bước đánh dấu những object còn sử dụng và những object không còn sử dụng.



viettel
software services

Mô hình hoạt động

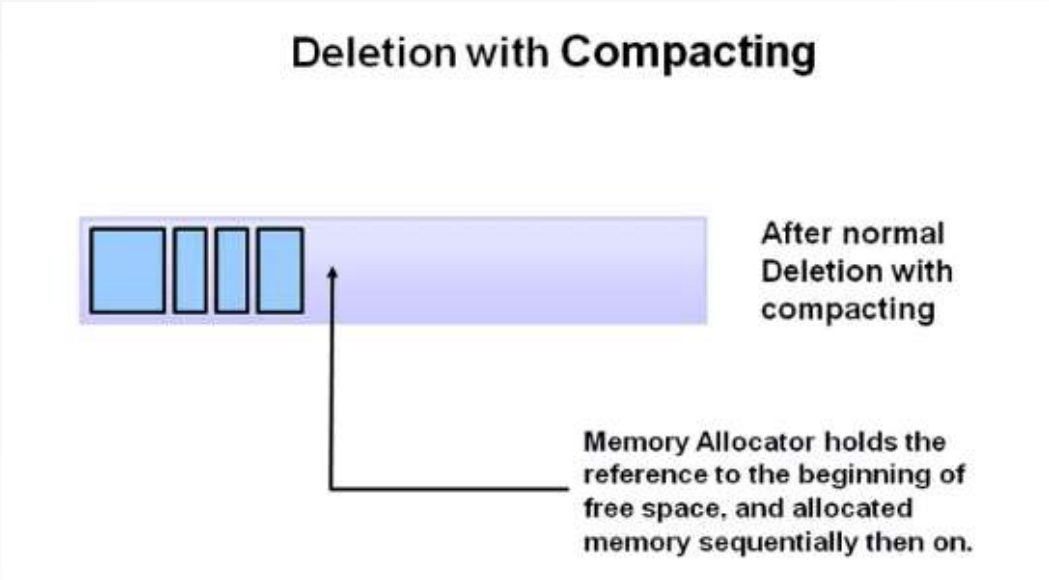
- **Bước 2 Normal deletion:** Trình Garbage Collection sẽ xóa các object không còn sử dụng.



viettel
software services

Mô hình hoạt động

- **Bước 3 Deletion with Compacting:** Sau khi những object không còn được sử dụng bị xóa, những object còn được sử dụng sẽ được gom lại gần nhau. Điều này giúp làm tăng hiệu suất sử dụng bộ nhớ trống để cấp phát cho những object mới.



viettel
software services

Cách triển khai

- Tuổi hoạt động của object được chia thành 3 nhóm tuổi: Young generation, old generation, và permanent generation.
- **Young generation:** nhóm này lại được chia thành 2 nhóm con là eden (khởi thủy) và survivor (sống sót). Nhóm survivor lại được chia thành 2 nhóm nhỏ hơn là S0 và S1. Các object mới được khởi tạo sẽ nằm trong nhóm Eden. Sau 1 chu kỳ hoạt động của garbage collector, object nào "sống sót" sẽ được chuyển sang nhóm survivor. Sự kiện các object ở nhóm Young generation được thu hồi bởi Garbage collector được xem là minor event.
- **Old generation:** nhóm này chứa các object chuyển từ young generation (tất nhiên với thời gian hoạt động đủ lâu, mỗi bộ garbage collector sẽ định nghĩa bao nhiêu được coi là "lâu"). Sự kiện các object ở nhóm Old generation được thu hồi bởi garbage collector được xem là major event.
- **Permanent generation:** nhóm này gồm metadata (ví dụ: các class, method....). Do đó, khi phải "dọn" các class, method không cần thiết, garbage collector sẽ tìm kiếm trong nhóm này. Java từ phiên bản JDK 8 trở đi thay vì sử dụng Permanent generation, Java sử dụng Metaspace để lưu trữ các metadata, bao gồm các class, method và biến tĩnh (static variable)

Cách sử dụng các bộ GC

- **Serial:** Các event được xử lý tuần tự trên 1 thread. Quá trình dồn bộ nhớ (compaction) được thực thi sau mỗi event. Serial là bộ garbage collector đơn giản nhất. Nó được thiết kế cho môi trường single thread. Khi Serial hoạt động, ứng dụng buộc phải dừng lại. Do đó nó không phù hợp với môi trường server. Để sử dụng thử Serial, dung lệnh -XX:+UseSerialGC khi chạy chương trình.
- **Parallel:** Parallel GC là trình dọn rác mặc định của JVM, không giống như Serial GC, Parallel GC được sử dụng trong nhiều thread để quản lý bộ nhớ. Minor event sẽ được xử lý trên nhiều thread. Major event và quá trình dồn bộ nhớ cho nhóm Old generation được xử lý trên 1 thread. Bên cạnh bộ Parallel, có một bộ khác tên là Parallel Old xử lý major event và quá trình dồn bộ nhớ (cho nhóm Old generation) trên nhiều thread. Nhược điểm của Parallel là khi hoạt động, nó sẽ dừng thread chạy chương trình

viettel
software services

Cách sử dụng các bộ GC

- **Concurrent Mark Sweep:** CMS xử lý event trên nhiều thread (giống với Parallel). Bên cạnh đó, CMS chạy song song với ứng dụng và đảm bảo quá trình dọn rác không làm ảnh hướng tới quá trình thực thi ứng dụng. CMS không tiến hành dồn bộ nhớ. So với 2 bộ garbage collector kể trên thì CMS tiêu tốn nhiều tài nguyên CPU hơn. Tuy nhiên nó lại không làm ảnh hưởng tới quá trình thực thi ứng dụng (hay còn gọi là trạng thái Stop The World - STW). Đối với các server hoặc các ứng dụng gặp bất lợi khi phải STW thì sử dụng CMS là lựa chọn phù hợp. CMS mặc định không hoạt động, ta cần bật nó lên bằng cách sử dụng tham số: XX:+UseConcMarkSweepGC.

viettel
software services

Cách sử dụng các bộ GC

- **Garbage first (G1):** Đây là bộ garbage collector mới nhất, ra đời cùng với Java 7 với mục tiêu thay thế cho CMS trong việc quản lý các vùng heap >4GB. G1 sử dụng nhiều background thread để scan qua vùng heap (được chia thành các vùng có dung lượng từ 1 đến 32MB). G1 sẽ thu dọn ở các vùng nhớ có nhiều "rác" nhất. So với CMS, G1 có khả năng vừa tiến hành thu hồi vừa dồn bộ nhớ (CMS chỉ có thể dồn bộ nhớ ở trong trạng thái STW). Mặc định, G1 không được bật, do đó ta cần sử dụng tham số: –XX:+UseG1GC.

    - Hơn nữa, G1 có khả năng nhận diện các chuỗi ký tự trùng nhau trong heap (được tham chiếu bởi các đối tượng khác nhau) và sửa tham chiếu nhằm tránh các bản copy thừa của các xâu ký tự, tiết kiệm dung lượng trống cho vùng nhớ heap. Để sử dụng tính năng String deduplication, ta truyền tham số: -XX:+UseStringDeduplication. String deduplication là một tính năng chung của JVM. Khi sử dụng cần cân nhắc để tránh lỗi hoặc mất dữ liệu

viettel
software services

Làm việc với GC hiệu quả

- Với các ứng dụng đơn giản, lập trình viên không cần quan tâm nhiều đến garbage collector. Tuy nhiên nếu họ muốn nâng level của bản thân thì việc hiểu cơ chế hoạt động và tùy chỉnh bộ garbage collector là một trong các kỹ năng phải học.
- Bên cạnh cơ chế hoạt động của garbage collector, các lập trình viên cần lưu ý một điều, đó là không thể dự đoán garbage collector sẽ chạy ở thời điểm nào. Kể cả khi ta có gọi nó một cách tường minh với System.gc() hay Runtime.gc(), ta vẫn không thể chắc chắn được garbage collector có chạy hay không.
- Về việc tùy chỉnh garbage collector, cách tiếp cận tốt nhất là điều chỉnh các setting flag của JVM (chính là các tham số tương ứng với từng bộ garbage collector được liệt kể ở trên), hoặc quy định kích thước khi cấp phát và kích thước tối đa của vùng nhớ heap mà chương trình sử dụng, hoặc điều chỉnh kích thước của từng nhóm "tuổi".

viettel
software services

BÀI 3

Tham chiếu và tham trị

viettel
software services

Tham chiếu và tham trị

- Khi gọi một phương thức và truyền một giá trị cho phương thức, được gọi là truyền tham trị. Việc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức. Trong Java, truyền tham trị dành cho các tham số có kiểu dữ liệu nguyên thủy là byte, short, int, long, float, double, boolean, char.
- Khi gọi một phương thức và truyền một tham chiếu cho phương thức, được gọi là truyền tham chiếu. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị của nó. Trong Java, tất các phương thức có tham số là biến có kiểu là các lớp (class) đều là kiểu tham chiếu.

viettel
software services

Tham trị

- Đoạn code xử lý như sau:

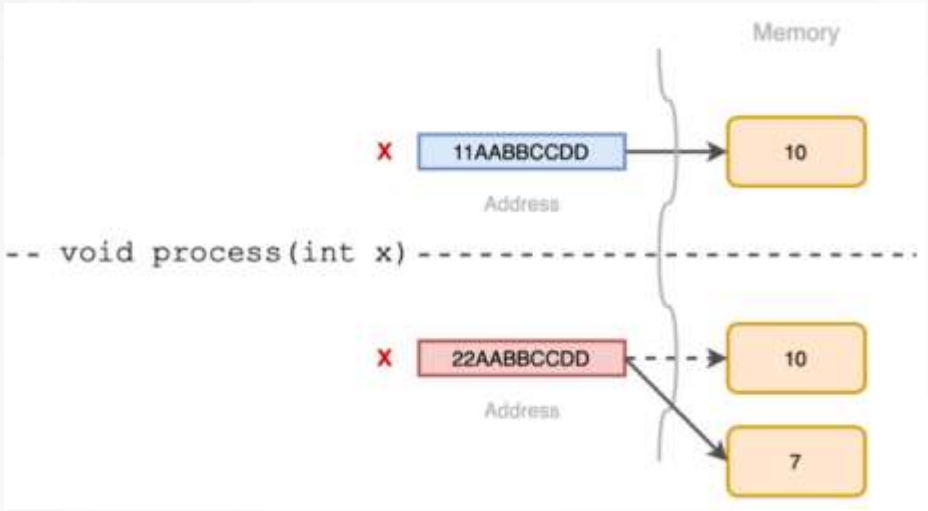
```
1.  public static void main(String[] args) {
2.      int x = 10;
3.
4.      System.out.println("Before call process: " + x);
5.      process(x);
6.      System.out.println("After call process: " + x);
7.  }
8.
9.  public static void process(int x) {
10.     x = 7;
11. }
```

viettel
software services

Tham trị

- Kết quả:

```
1. Before call process: 10
2. After call process: 10
```



viettel
software services

Tham chiếu

- Tạo class:

```
1.  public class MyCat {
2.      private String name;
3.
4.      public MyCat(String name) {
5.          this.name = name;
6.      }
7.
8.      public String getName() {
9.          return name;
10.     }
11.
12.     public void setName(String name) {
13.         this.name = name;
14.     }
15. }
```

viettel
software services

Tham chiếu

- Hàm main xử lý đoạn code như sau:

```
1.  public static void main(String[] args) {
2.      MyCat myCat = new MyCat("Kitty");
3.
4.      System.out.println("Before call process: " + myCat.getName());
5.      process(myCat);
6.      System.out.println("After call process: " + myCat.getName());
7.  }
8.
9.  public static void process(MyCat myCat) {
10.     myCat.setName("Doraemon");
11. }
```

viettel
software services

Tham chiếu

- Hàm main xử lý đoạn code như sau:

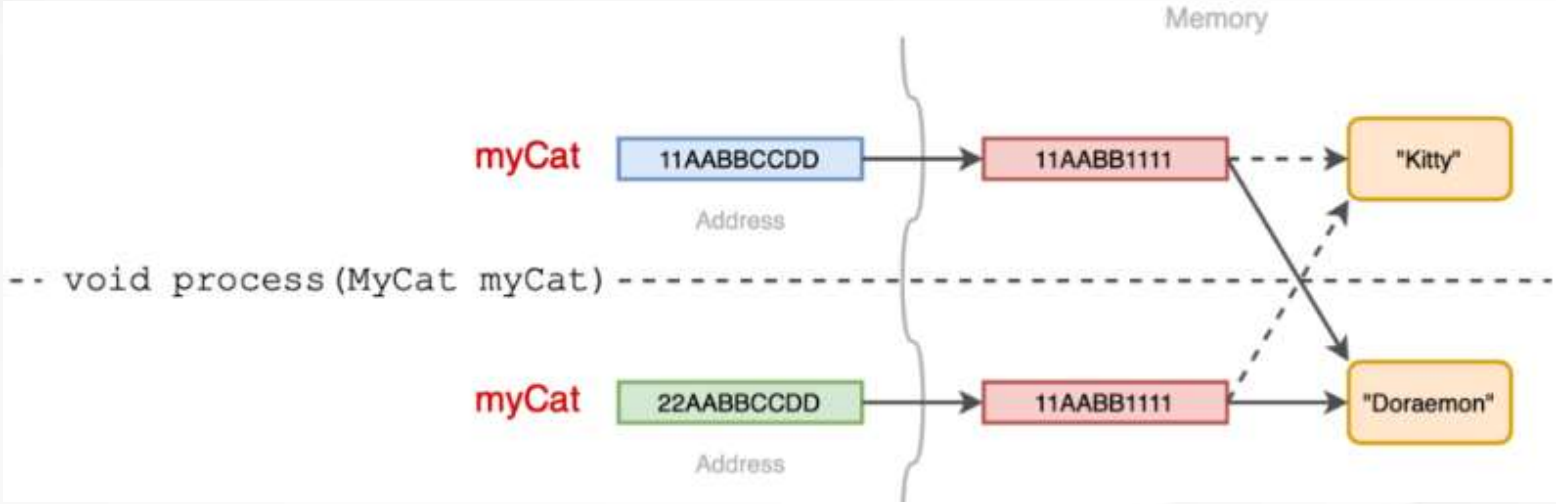
```
1.  public static void main(String[] args) {
2.      MyCat myCat = new MyCat("Kitty");
3.
4.      System.out.println("Before call process: " + myCat.getName());
5.      process(myCat);
6.      System.out.println("After call process: " + myCat.getName());
7.  }
8.
9.  public static void process(MyCat myCat) {
10.     myCat.setName("Doraemon");
11. }
```

viettel
software services

Tham chiếu

- Kết quả:

```
1. Before call process: Kitty
2. After call process: Doraemon
```



viettel
software services

XIN CẢM ƠN