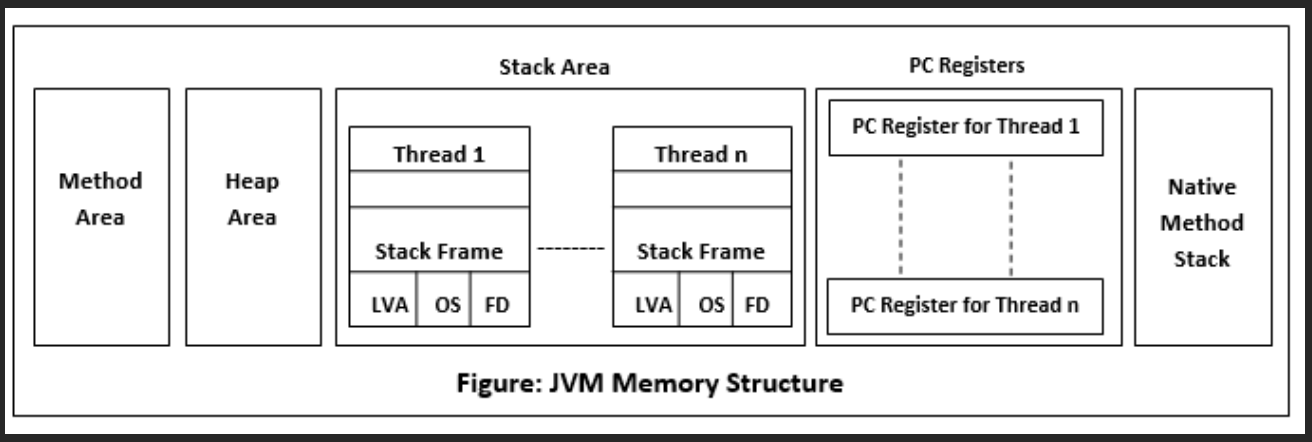
Adresování a správa paměti

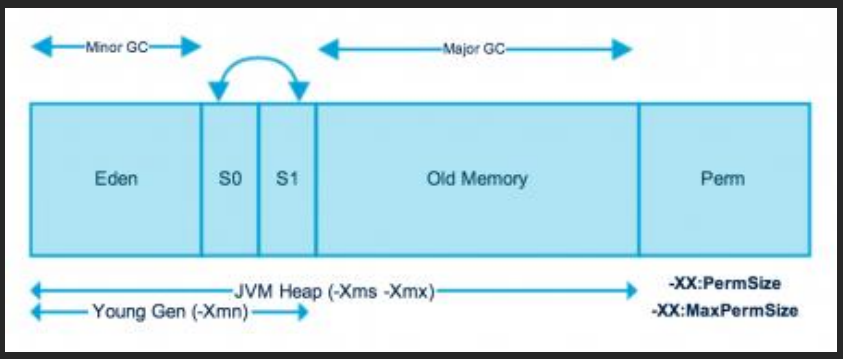
V Javě se o správu paměti starat nemusíme, *JVM (*Java virtual Machine) to dělá za nás. I přesto ale si musíme dávat pozor na objekty, které vytváříme.

Java funguje na principu **zásobníku** (***Stack memory***). Každé vlákno má přesně jeden zásobník.

Virtuální paměť v Javě se dělí na několik částí – *Method Area, Heap Space, Stack, Registry a Native Method Stack.*



- Java Virtual Machine Memory Structure



- *Java Heap Model*

Method Area

Method Area je část paměti, ve které se ukládají informace o třídách, metodách a interfacích a jedná se o místo, odkud jsou třídy načítány. Je zde uložen binární kód tříd, statické a globální proměnné, jaké třídy implementují jaké interface a konstanty a String literals (Je zde String pool)

String Pool

String pool je místo, na kterém se ukládají String, které nejsou vytvořené pomocí keywordu „new“. Pokud vytváříme dva stringy se stejným obsahem, vytvoří se pouze jeden a druhý bude obsahovat referenci na onen první. Příklad:

```
public String ahoj = „ahoj“; //Vytvoří se string literal
public String ahoj = „ahoj“; //Bude uchovávat referenci na string
literal
```

Heap space

Heap space je nejznámější část. Zabírá největší část RAMky. Ukládají se zde všechny objekty, které se během vývoje vytvářejí. Obsahuje také všechny objekty, které vlákna sdílejí.

Heap se také dělí na více částí –Young generation a Old generation. Každý objekt se mezi těmito části přesouvá.

Young Generatior (YG)

Má dvě části : Eden a Survivor space (Ten má dva – S0 a S1)
Nově vytvořené objekty jsou umístěny v Edenu.
Jakmile se YG zaplní, proběhne Garbage collection. Pokud je objekt používán dostatečně dlouho, přesouvá se do old generation.

V survivor space jsou objekty, které přežily Minor Garbage Collecting (Minor = Jenom YG, Major je OG).

Old Generation(OG)

OG paměť obsahuje dlouho žijící objekty, které přežily několik Garbage Collectingů / určitou věkovou hranic. Garbage Collecting se v OG dělá když je paměť plná a většinou trvá dlouho. Má pouze jednu část - Tenured Generation.

Perm Gen(PG)

PG uchovává informace o metadatech (Data, která ukazují na další data), které JVM vyžaduje k popisu tříd a metod, které se používají. Příklad může být deklarace třídy, metody nebo pole. Perm Gen je součástí Heapu. Zkratka značí „Permanent Generation“.

Native Method Stack

Native Method Stack je část programu, která se zaměřuje na spouštění programů, které jsou psány v jiných jazycích než v Javě. Je oddělený od jiných stacků.

Meta Space(MeSp)

Metaspace přichází s nástupem Javy 8.0. Do té doby byl používán právě PermGen. Meta space nemá žádný vrchní limit. Má se vyvarovat OutOfMemorySpace erroru. Pokud ovšem přerostě ve více než je naše fyzická RAM paměť, začne využívát virtuální. To je ovšem velmi náročná a drahá operace.

Stack Memory

Zásobníková paměť fungující na principu Last in, First out (LIFO). Jsou na ní uloženy frames – ty uchovávají informace při spuštění metody(Například o parametrech a lokální proměnné), reference na objekty na Heapu a také primitivní proměnné (int, bool, double).

PC Registry

Jsou využívání k ukládání adres instrukce, která se aktuálně provádí. Instrukce je příkaz pro procesor a registry instrukce ukládají k tomu, aby se provedly ve správném pořadí. Když se spustí nová metoda, vytvoří se instrukce a ty se uloží do registrů podle potřebného pořadí. Řeší i výjimky, protože ukazují na poslední instrukci, která byla provedena.

Garbage collecting

Garbage collection (GC) je proces, ve kterém se **zbavujeme nepotřených dat** zabírající místo v paměti, protože se nám může stát například „OutOfMemoryError“, kdy programu dojde paměť.

GC v Javě funguje **automaticky.** Jedná se o proces v heap paměti, které identifikuje objekty, které se používají a které ne, načež je smaže.

Nepoužívaný objekt / Nedosažitelný objekt = **Nemá** na sobě žádnou **referenci** v jakékoliv části programu.
Používaný objekt = objekt, na který je reference či pointer

Programátor v Javě nemusí tyto objekty nijak označovat, **implementace** GC **žije** v *JVM*.

Garbage Collecting má **dva** typy:

Minor

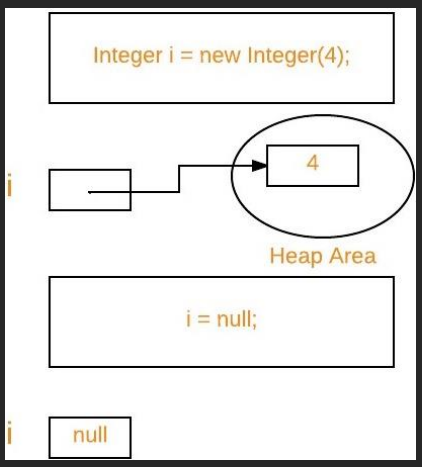
Minoritní GC se stane v momentě, když je **odebrán nedosažitelný** / nepoužívaný objekt nebo je **pamět plná**. Příklad:

```
Integer i = new Integer(4);

// Objekt je nyní dosažitelný přes
referenci „i“

i = null;

// Objekt je nyní nedosažitelný.
```



Major

Má se stát v momentě, kdy objekty **přežijí** Minor *GC* a jsou **zkopírovány** do *OG*. Jak můžeme logicky vidět, **stává se méně často** a je pouze pro Old Generation. Trvá **déle**, jelikož prohledává všechny dlouho žijící objekty.

Metoda zavolání GC

```
1.)      System.gc()

2.)      Runtime.getRuntime().gc();
```

Jak probíhá GC?

GC se zavolá když je paměť v Edenu (YG) plná

Podívá se na prvky a ty, které jsou referencované či opointerované se přenesou do S0. Zbylé se smažou

Při dalším GC se stane s Edenem to samé, ale všechny prvky se přesunou do S1 – i ty, které jsou v S0 z minulého a smaže se celá S0

Pokud takto prvky přeskočí 8krát – Říká se tomu aging(stárnutí), jsou přesunuty do Old Generation -> Do jediného místa, Tenured Generation

Reference a pointery

Reference je adresa, která odkazuje, kde je metoda, objekt, proměnná v paměti uchována. Pointer je v Javě to samé.

Pokud totiž vytvoříme objekt, udělá se nám v Heap paměti místo. A díky referencím se k tomuto místu můžeme dostat.

Referenční proměnné

Referenční proměnné jsou používány k získání objektů či hodnot.

Pokud například vytváříme instanci objektu, tak ona instance je referenční proměnná – Protože ukazuje na místo v paměti, kde je objekt uchováván.

Regerence jsou ukládány na Stacku

Trida t1 = new Trida(); -> t1 je referenční proměnná