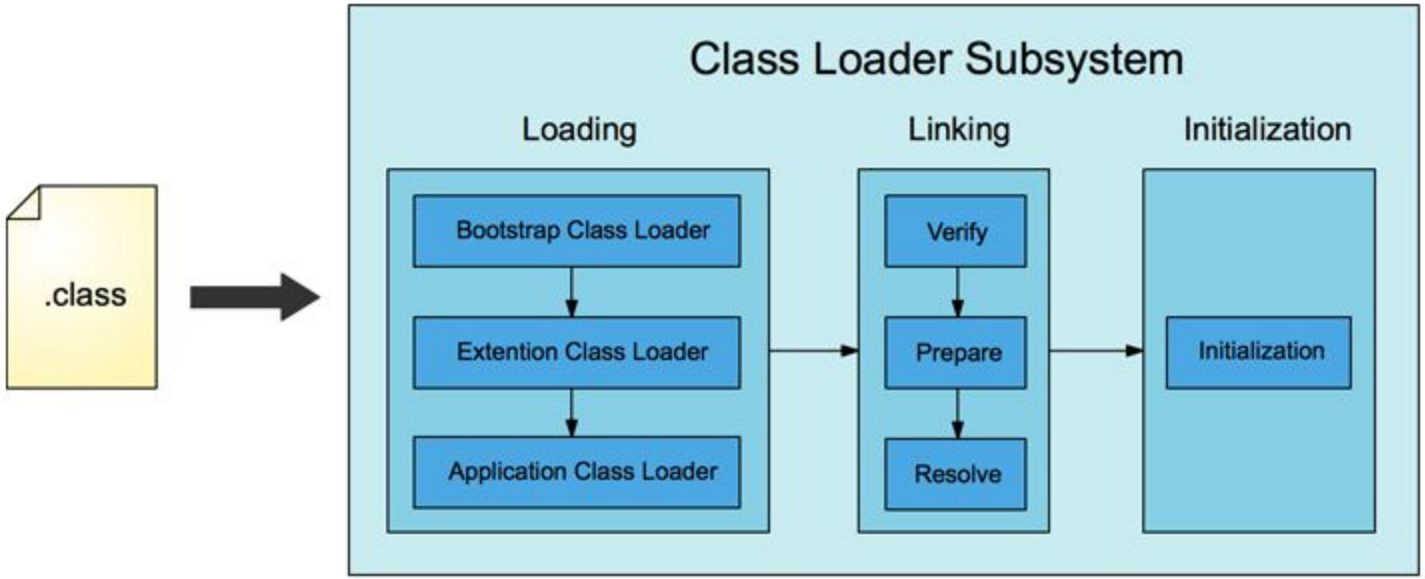
Lesson 18

03.07.2023


.class
Class Loader Subsystem
Loading
Linking
Initialization
Bootstrap Class Loader
Extention Class Loader
Application Class Loader
Verify
Prepare
Resolve
Initialization

Види завантажувачів

Розрізняють три види завантажувачів в Java. Це базовий завантажувач (**bootstrap**), системний завантажувач (**System Classloader**), завантажувач розширень (**Extension Classloader**).

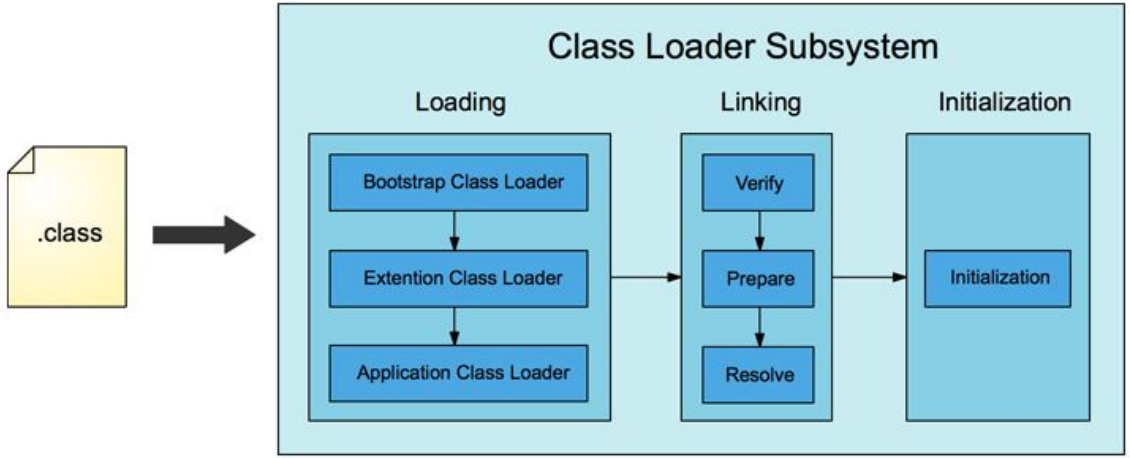
Bootstrap - реалізований на рівні JVM і не має зворотного зв'язку із середовищем виконання. Завантажувачем завантажуються класи з директорії $JAVA_HOME/lib. Тобто. всіма улюблений rt.jar завантажується саме базовим завантажувачем. Тому, спроба отримання завантажувача класів java.* завжди закінчуються null'ом. Це тим, що це базові класи завантажені базовим завантажувачем, доступу до якого з керованого середовища немає.

Керувати завантаженням базових класів можна за допомогою ключа -Xbootclasspath, який дозволяє перевизначати набори базових класів.

System Classloader - системний завантажувач, реалізований вже на рівні JRE. У Sun JRE - це клас sun.misc.Launcher $ AppClassLoader. Цим завантажувачем завантажуються класи, шляхи яких зазначені в змінної оточення CLASSPATH.

Керувати завантаженням системних класів можна за допомогою ключа -classpath або системною опцією java.class.path.

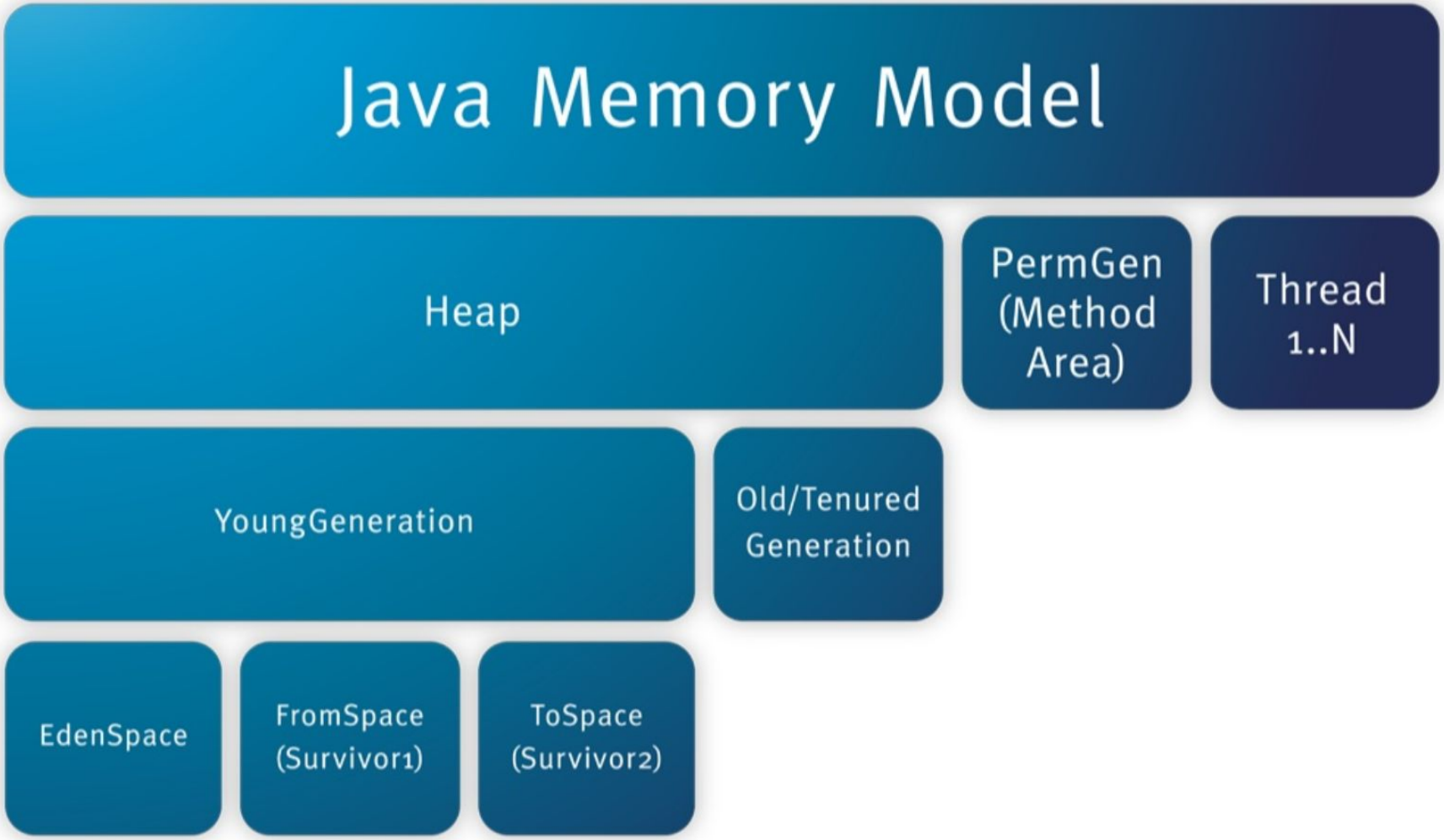
Extension Classloader – завантажувач розширень. Цей завантажувач завантажує класи з директорії $JAVA_HOME/lib/ext. У Sun JRE - це клас sun.misc.Launcher $ ExtClassLoader.

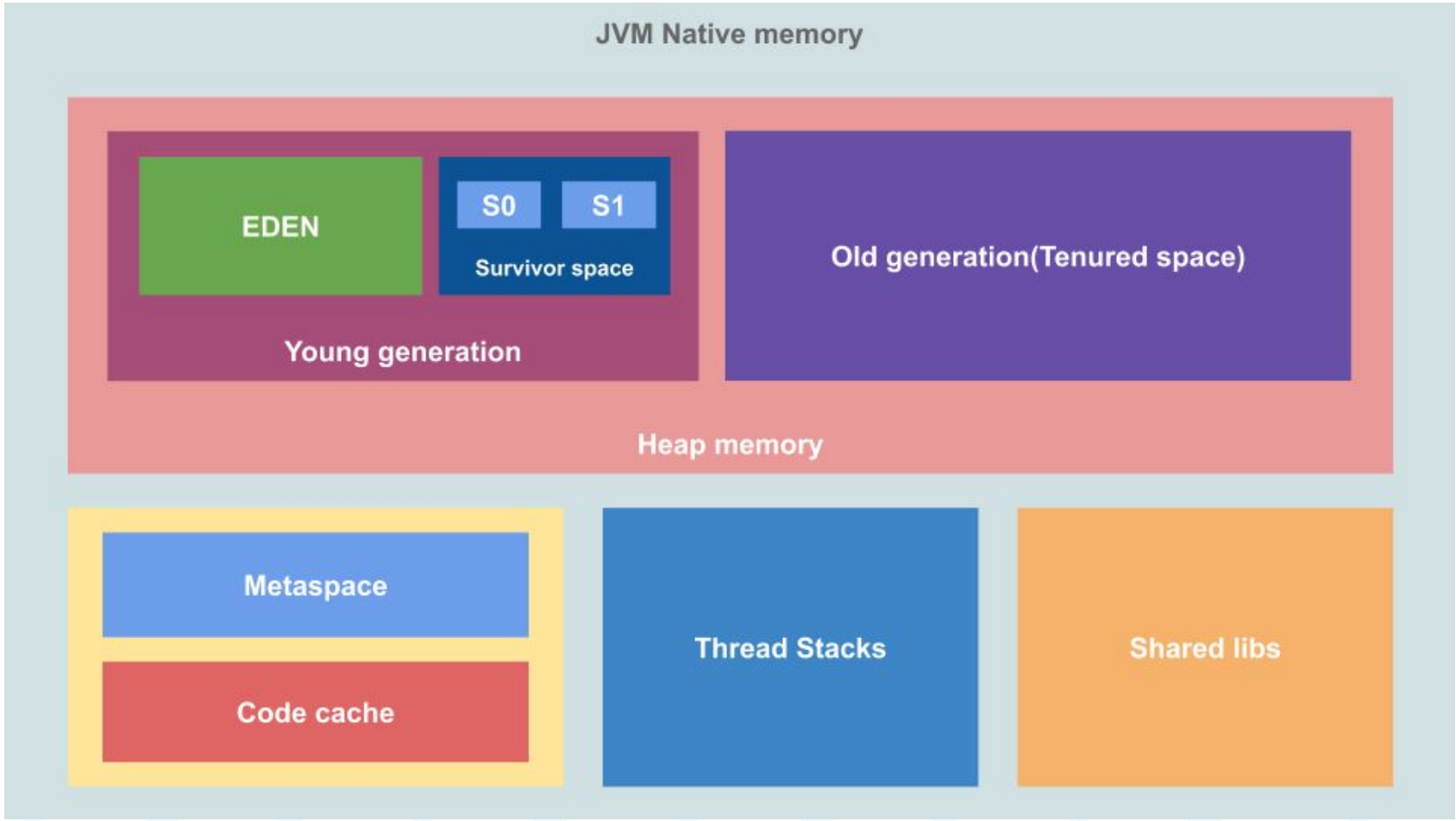


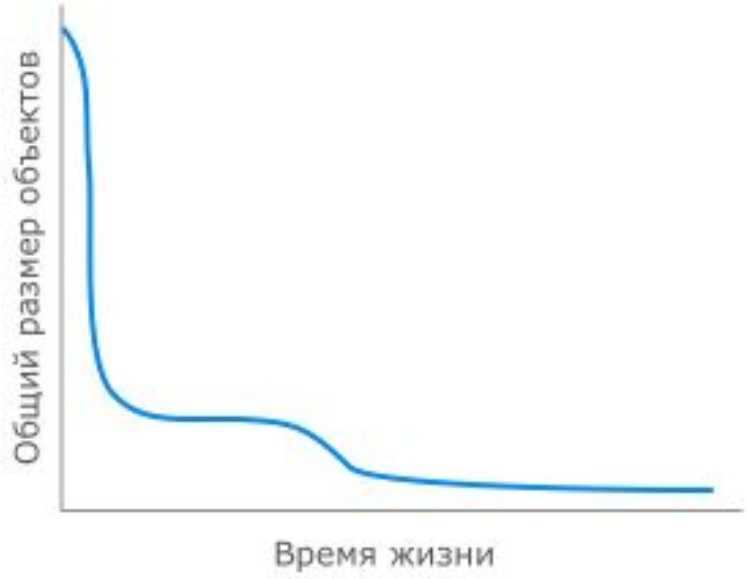
Loading – на цій фазі відбувається пошук та фізичне завантаження файлу класу у певному джерелі (залежно від завантажувача). Цей процес визначає базове уявлення класу у пам'яті. У цьому етапі такі поняття як методи, поля тощо. поки що не відомі.

Linking - процес, який може бути розбитий на 3 частини:
1. Bytecode verification – відбувається кілька перевірок байт-коду на відповідність ряду найчастіше нетривіальних вимог, визначених у специфікації JVM.
2. Class preparation – на цьому етапі відбувається підготовка структури даних, що відображає поля, методи та реалізовані інтерфейси, які визначені в класі.
3. Resolving – дозвіл всіх класів, які посилаються на поточний клас.

Initialization – відбувається виконання статичних ініціалізаторів, визначених у класі. Отже, статичні поля ініціалізуються стандартними значеннями.


Java Memory Model
Heap
PermGen (Method Area)
Thread 1..N
YoungGeneration
Old/Tenured Generation
EdenSpace
FromSpace (Survivor1)
ToSpace (Survivor2)


JVM Native memory
EDEN
S0
S1
Survivor space
Young generation
Old generation(Tenured space)
Heap memory
Metaspace
Code cache
Thread Stacks
Shared libs

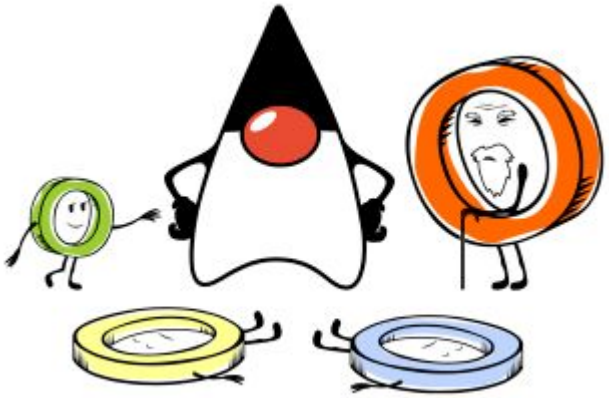


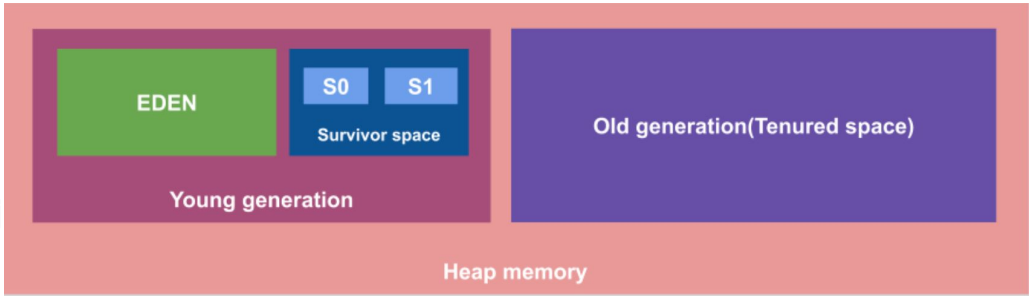
Переважна більшість об'єктів створюються **на дуже короткий час**, вони стають непотрібними практично відразу після першого використання. Ітератори, локальні змінні методів, результати боксингу та інші тимчасові об'єкти, які найчастіше створюються неявно, потрапляють саме до цієї категорії, утворюючи пік на самому початку графіка.

Далі йдуть об'єкти, що створюються для виконання **більш-менш тривалих обчислень**. Їхнє життя трохи різноманітніше — вони зазвичай гуляють різними методами, трансформуючись і збагачуючись у процесі, але після цього стають непотрібними і перетворюються на сміття. Завдяки таким об'єктам виникає невеликий горбок на графіку слідом за піком тимчасових об'єктів.

І, нарешті, **об'єкти-старожили**, які переживають майже всіх — це постійні дані програми, які часто завантажуються на самому початку і проживають довге і щасливе життя до зупинки програми.

Ось тут і виникає ідея поділу об'єктів на **молодше покоління (young generation)** та **старше покоління (old generation)**. Відповідно до цього поділу і процеси складання сміття поділяються **на малу збірку (minor GC), що стосується тільки молодшого покоління, і повну збірку (full GC), яка може зачіпати обидва покоління**. Малі складання виконуються досить часто і видаляють основну частину мертвих об'єктів. Повні збирання виконуються тоді, коли поточний обсяг виділеної програмі пам'яті близький до вичерпання і малою збиранням вже не обійтися.





Ефективність роботи GC:

- Максимальна затримка – максимальний час, на який GC припиняє виконання програми для виконання однієї збірки. Такі зупинки називаються stop-the-world (або STW).
- Пропускна здатність - відношення загального часу роботи програми до загального часу простою, спричиненого складання сміття, на тривалому проміжку часу.
- Споживані ресурси - обсяг ресурсів процесора та/або додаткової пам'яті, що споживаються збирачем.

Garbage Collector виконує всього два завдання, пов'язані з пошуком сміття та його очищенням. Для виявлення сміття існує два підходи

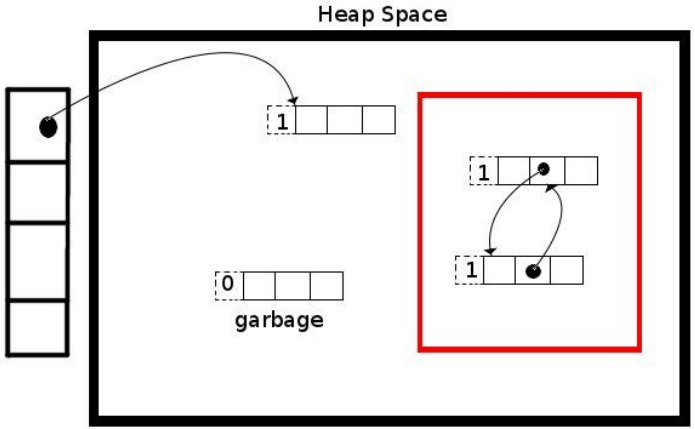
Reference counting – облік посилань;

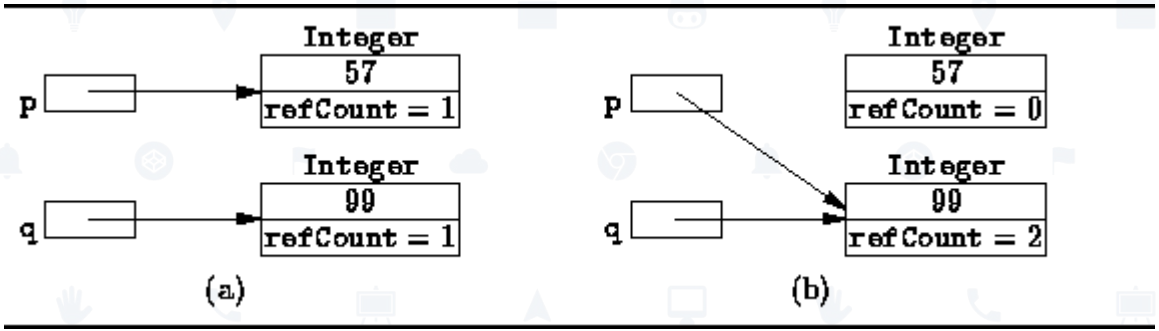
Tracing – трасування.

Reference counting

Суть підходу «**Reference counting**» пов'язана з тим, що кожен об'єкт має лічильник, який зберігає інформацію про кількість посилань, що вказують на нього. При знищенні посилання лічильник зменшується. При нульовому значенні лічильника об'єкт вважатимуться сміттям.

Головним недоліком даного підходу є складність забезпечення точності лічильника та «неможливість» виявляти циклічні залежності. Так, наприклад, два об'єкти можуть посилатися один на одного, але на жоден з них немає зовнішнього посилання. Це супроводжується витоком пам'яті. У зв'язку з цим даний підхід не набув поширення.

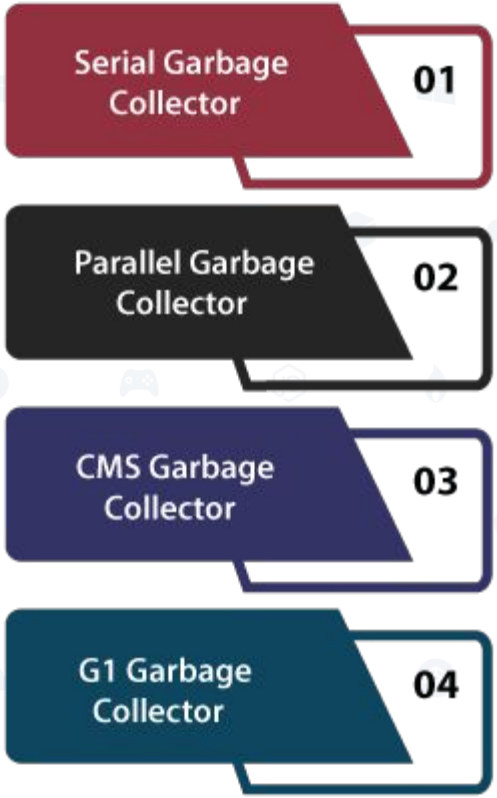




Tracing

Головна ідея Tracing пов'язана з тим, що до живого об'єкта можна дістатися з кореневих точок (GC Root). Все, що є з «живого» об'єкта, також є «живим». Якщо представити всі об'єкти та посилання між ними як дерево, необхідно пройти від кореневих вузлів GC Roots по всіх вузлах. При цьому вузли, до яких не можна дістатися, є сміттям.

Types of Garbage Collector in Java



Serial (послідовний) – найпростіший варіант для додатків з невеликим обсягом даних та не вимогливих до затримок. Рідко коли використовується, але на слабких комп'ютерах може бути обраний віртуальною машиною як збирач за замовчуванням.

Parallel(паралельний) — успадковує підходи до збирання від послідовного збирача, але додає паралелізм в деякі операції, а також можливості автоматичного підстроювання під необхідні параметри продуктивності.

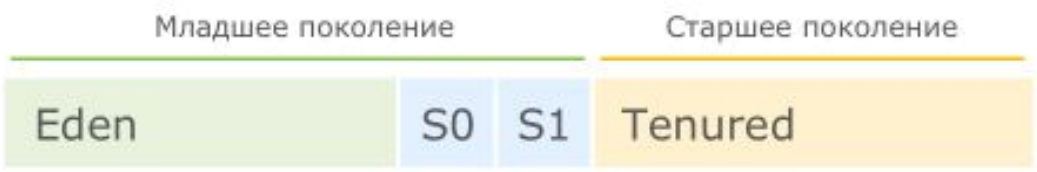
Concurrent Mark Sweep (CMS) - націлений на зниження максимальних затримок шляхом виконання частини робіт зі складання сміття паралельно з основними потоками програми. Підходить до роботи з відносно великими обсягами даних у пам'яті.

Garbage-First (G1) — створений для поступової заміни CMS, особливо в серверних програмах, що працюють на багатопроцесорних серверах і оперують великими обсягами даних.

Serial Garbage Collector 01

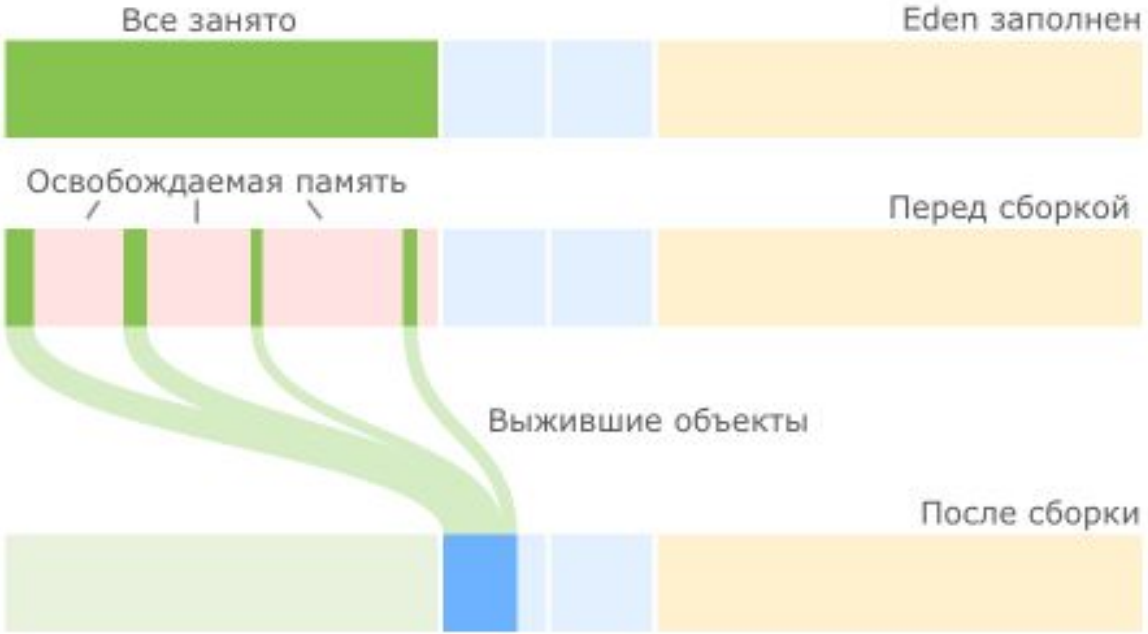
Використання Serial GC включається **-XX:+UseSerialGC**.

При використанні даного збирача купа розбивається на чотири регіони, три з яких відносяться до молодшого покоління (Eden, Survivor 0 та Survivor 1), а один (Tenured) до старшого:



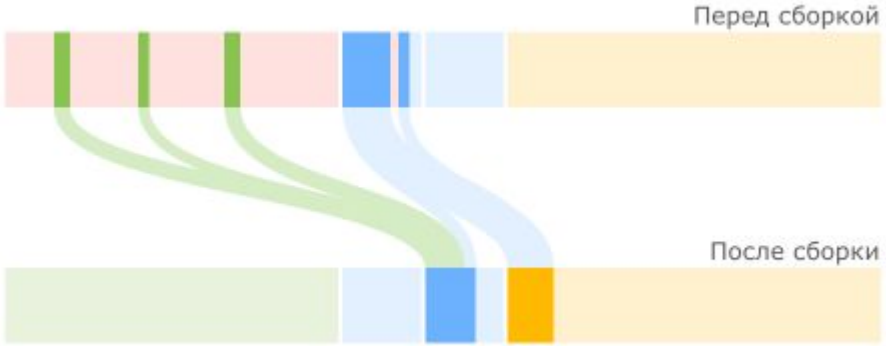
Середньостатистичний об'єкт починає своє життя у регіоні Eden. Саме сюди його поміщає JVM у момент створення. Але з часом може виявитися так, що місця для новоствореного об'єкта в Eden немає, у таких випадках запускається малий GC.

По перше GC знаходить і видаляє мертві об'єкти з Eden. Живі об'єкти, що залишилися, переносяться в порожній регіон Survivor. Один із двох регіонів Survivor завжди порожній, саме він вибирається для перенесення об'єктів з Eden:



Після малогї збірки сміття регіон Eden повністю випорожнений і може бути використаний для розміщення нових об'єктів. Але рано чи пізно наша програма знову займе всю область Eden і JVM знову спробує провести малу збірку, цього разу очищаючи Eden і частково зайнятий Survivor 0, після чого переносячи всі об'єкти, що вижили, в порожній регіон Survivor 1:

JVM постійно стежить за тим, як довго об'єкти переміщуються між Survivor 0 і Survivor 1, і вибирає відповідний поріг для кількості таких переміщень, після якого об'єкти переміщуються в Tenured, тобто переходять у старше покоління. Якщо регіон Survivor виявляється заповненим, то об'єкти з нього також відправляються до Tenured:



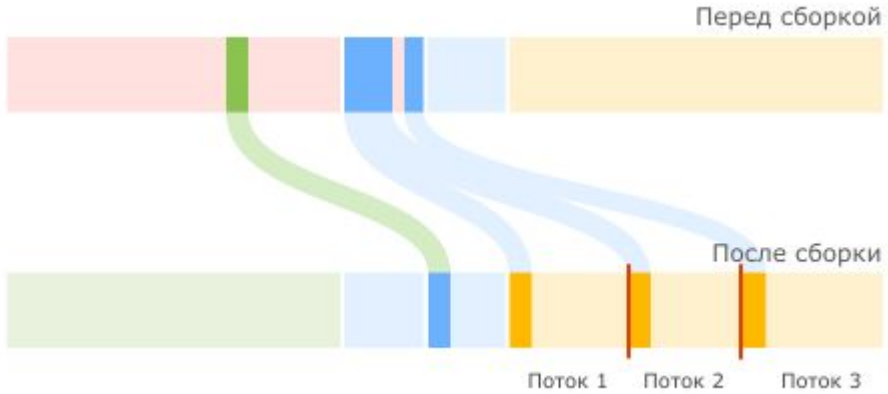
У випадку, коли місця для нових об'єктів не вистачає вже в Tenured, у справу вступає повне прибирання сміття, що працює з об'єктами обох поколінь. При цьому старше покоління не ділиться на підрегіони за аналогією з молодшим, а є одним великим шматком пам'яті, тому після видалення мертвих об'єктів з Tenured проводиться не перенесення даних (переносити вже нікуди), а їх ущільнення, тобто розміщення послідовно, без фрагментації. Такий механізм очищення називається Mark-Sweep-Compact за назвою його кроків (позначити об'єкти, що вижили, очистити пам'ять від мертвих об'єктів, ущільнити вижили об'єкти).

З цим збирачем все досить просто, тому що вся його робота – це один суцільний STW. На початку кожного складання сміття робота основних потоків програми зупиняється і відновлюється лише після закінчення складання. Причому всю роботу з очищення Serial GC виконує не кваплячись, в одному потоці, послідовно, за що й удостоївся свого імені.

Parallel Garbage Collector 02

Паралельний збирач включається **-XX:+UseParallelGC.**

При підключенні паралельного збирача використовуються ті ж підходи до організації купи, що і у випадку з Serial GC - вона ділиться на такі ж регіони Eden, Survivor 0, Survivor 1 і Old Gen, що функціонують за тим же принципом. Але є дві принципові відмінності у роботі з цими регіонами: по-перше, збиранням сміття займаються **кілька потоків паралельно**; по-друге, цей збирач може **самостійно підлаштовуватися під необхідні параметри продуктивності**. Давайте розберемося, як це влаштовано.



За умовчанням і мале і повна збірка сміття задіють багатопоточність. Мала користується нею під час перенесення об'єктів у старше покоління, а повна — за ущільнення даних у старшому поколінні.

Кожен потік збирача отримує свою ділянку пам'яті в регіоні Old Gen, так званий буфер підвищення (promotion buffer), куди він може переносити дані, щоб не заважати іншим потокам.

Ситуації STW

Як і у випадку з послідовним збирачем, на час операцій з очищення пам'яті всі основні потоки програми зупиняються. Різниця лише в тому, що пауза, як правило, коротша за рахунок виконання частини робіт у паралельному режимі.

CMS Garbage Collector 03

Використання CMS GC включається опцією **-XX:+UseConcMarkSweepGC**

Ми вже зустрічали слова Mark та Sweep при розгляді послідовного та паралельного збирачів. Вони позначали два кроки в процесі складання сміття у старшому поколінні: позначку об'єктів, що вижили, і видалення мертвих об'єктів. Складальник CMS отримав свою назву завдяки тому, що виконує ці кроки паралельно з роботою основної програми.

При цьому CMS GC використовує ту ж саму організацію пам'яті, що і вже розглянуті Serial/Parallel GC: регіони Eden+Survivor 0+Survivor 1+Tenured і такі ж принципи малого складання сміття. Відмінності починаються лише тоді, коли справа доходить до повного складання. У разі CMS її називають старшою (major) збіркою, а не повною, оскільки вона не торкається об'єктів молодшого покоління. В результаті, малі та старші зборки тут завжди розділені

Починається вона із зупинки основних потоків програми та позначки всіх об'єктів, безпосередньо доступних з коренів. Після цього додаток відновлює свою роботу, а збирач паралельно з ним здійснює пошук усіх живих об'єктів, доступних за посиланнями з тих самих помічених кореневих об'єктів (цю частину він робить в одному або кількох потоках).

Звичайно, за час такого пошуку ситуація в купі може змінитися, і не вся інформація, зібрана під час пошуку живих об'єктів, виявляється актуальною. Тому збирач ще раз припиняє роботу програми та переглядає купу для пошуку живих об'єктів, що вислизнули від нього за час першого проходу.

Після того, як живі об'єкти позначені, робота основних потоків програми відновлюється, а збирач здійснює очищення пам'яті від мертвих об'єктів у кількох паралельних потоках. При цьому слід мати на увазі, що після очищення не проводиться упаковка об'єктів у старшому поколінні, оскільки робити це при додатку, що працює, дуже важко.

Ситуації STW

З усього сказаного вище випливає, що при звичайній складання сміття у CMS GC існують такі ситуації, що призводять до STW:

Мала збірка сміття. Ця пауза нічим не відрізняється від аналогічної паузи у Parallel GC.

Початкова фаза пошуку живих об'єктів при старшому збиранні (так звана initial mark pause). Ця пауза зазвичай дуже коротка.

Фаза доповнення набору живих об'єктів при старшому збиранні (відома також як remark pause). Вона зазвичай довша за початкову фазу пошуку.

У разі виникнення збою конкурентного режиму пауза може затягтися досить тривалий час.

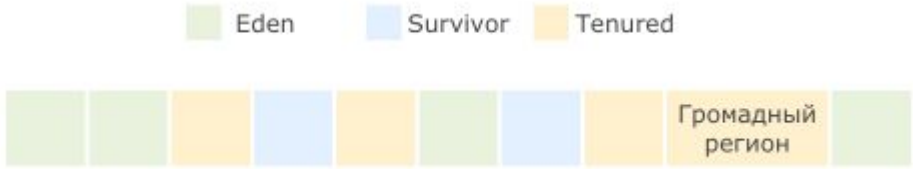
G1 Garbage Collector 04

G1 включається опцією Java-XX: **+ UseG1GC**.

G1 позиціонується як збирач для додатків з великими купами (від 4 ГБ і вище), для яких важливо зберігати час відгуку невеликим і передбачуваним, навіть за рахунок зменшення пропускної здатності.

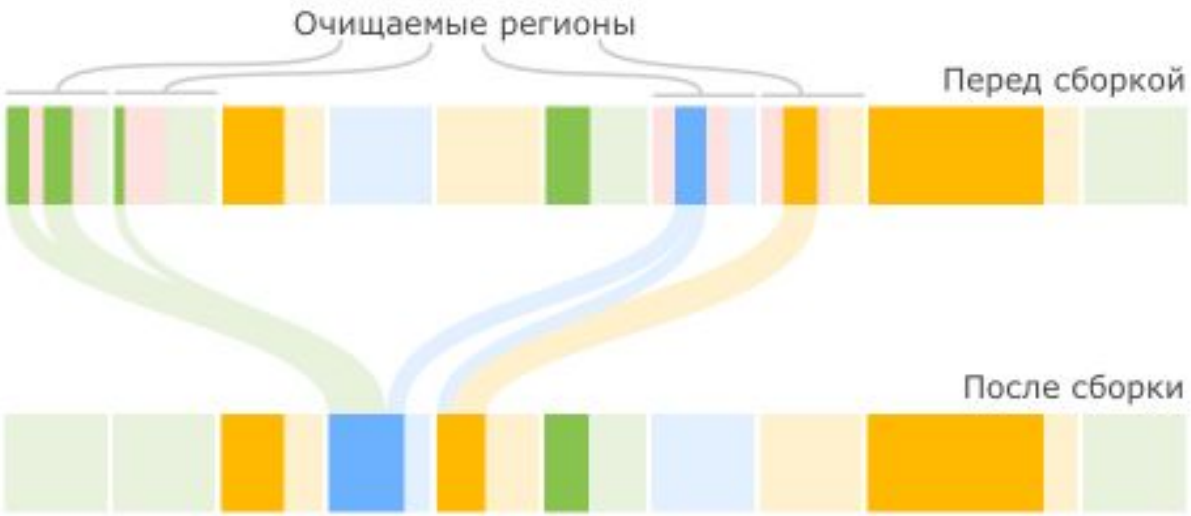
Перше, що впадає у вічі під час розгляду G1 — це зміна підходи до організації купи. Тут пам'ять розбивається на множину регіонів однакового розміру. Розмір цих регіонів залежить від загального розміру купи і за замовчуванням вибирається так, щоб їх було не більше ніж 2048, зазвичай виходить від 1 до 32 МБ. Виняток становлять лише звані величезні (humongous) регіони, які створюються об'єднанням традиційних регіонів розміщувати дуже великих об'єктів.

Розподіл регіонів на Eden, Survivor і Tenured у разі логічний, регіони одного покоління не повинні йти поспіль і навіть можуть змінювати свою приналежність до того чи іншого покоління. Приклад поділу купи на регіони може виглядати наступним чином (кількість регіонів сильно зменшена):



Малі зборки виконуються періодично для очищення молодшого покоління та перенесення об'єктів у регіони Survivor, або їх підвищення до старшого покоління з перенесенням у Tenured. Над перенесенням об'єктів працюють кілька потоків, і тимчасово цього процесу робота основного докладання зупиняється. Це вже знайомий нам підхід із розглянутих раніше збирачів, але відмінність полягає в тому, що очищення виконується не на всьому поколінні, а лише на частині регіонів, які збирач зможе очистити не перевищуючи бажаного часу. При цьому він обирає для очищення ті регіони, в яких, на його думку, накопичилася найбільша кількість сміття та очищення яких дасть найбільший результат. Звідси саме назва Garbage First — сміття насамперед.

А з повним очещенням (точніше, тут воно називається змішаним (mixed)) все трохи хитромудріше, ніж у розглянутих раніше збирачах. У G1 існує процес, званий циклом позначки (marking cycle), який працює паралельно з основним додатком та складає список живих об'єктів. Крім останнього пункту, цей процес виглядає вже знайомо для нас:
Initial mark. Позначка коренів (зі зупинкою основної програми) з використанням інформації, отриманої з малих складання.
Concurrent marking. Позначка всіх живих об'єктів у кількох потоках, паралельно з роботою основноъ програми.
Remark. Додатковий пошук неврахованих раніше живих об'єктів (зі зупинкою основної програми).
Cleanup. Очищення допоміжних структур обліку посилань на об'єкти та пошук порожніх регіонів, які можна використовувати для розміщення нових об'єктів. Перша частина цього кроку виконується при зупиненому основному додатку.


Очищаемые регионы
Перед сборкой
После сборки

Ситуації STW

Якщо резюмувати, то у G1 ми отримуємо STW у таких випадках:

Процеси перенесення об'єктів між поколіннями. Для мінімізації таких пауз G1 використовує кілька потоків.

Коротка фаза початкової позначки коренів у рамках циклу позначки.

Більш довга пауза в кінці фази remark і на початку фази cleanup циклу позначки.

Паттерн проектування - це вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.
На відміну від готових функцій або бібліотек, патерн не можна просто взяти та скопіювати у програму. Паттерн є не якимось конкретним кодом, а загальною концепцією вирішення тієї чи іншої проблеми, яку потрібно буде ще підлаштувати під потреби вашої програми.
Паттерни часто плутають із алгоритмами, адже обидва поняття описують типові рішення якихось відомих проблем. Але якщо алгоритм – це точний набір дій, то патерн – це високорівневий опис рішення, реалізація якого може відрізнятись у двох різних програмах.
Якщо навести аналогії, то алгоритм – це кулінарний рецепт із чіткими кроками, а патерн – інженерний креслення, на якому намальовано рішення, але не конкретні кроки його реалізації.

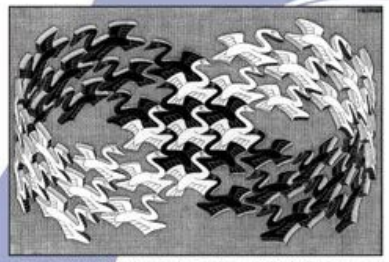
Концепцію патернів вперше описав Крістофер Александер у книзі «Мова шаблонів. Міста. Будинки. Будівництво». У книзі описано «мову» для проектування навколишнього середовища, одиниці якого — шаблони (або патерни, що ближче до оригінального терміну patterns) — відповідають на архітектурні питання: якої висоти зробити вікна, скільки поверхів має бути в будівлі, яку площу у мікрорайоні відвести під дерева та газони.

Ідея здалася привабливою четвірці авторів: Еріху Гамме, Річарду Хелму, Ральфу Джонсону, Джону Вліссідесу. 1995 року вони написали книгу «Прийоми об'єктно-орієнтованого проектування. Паттерни проектування», до якої увійшли 23 патерни, які вирішують різні проблеми об'єктно-орієнтованого дизайну. Назва книги була занадто довгою, щоб хтось зміг всерйоз її запам'ятати. Тому невдовзі всі почали називати її "book by the gang of four", тобто "книга від банди чотирьох", а потім взагалі "GoF book".

Design Patterns

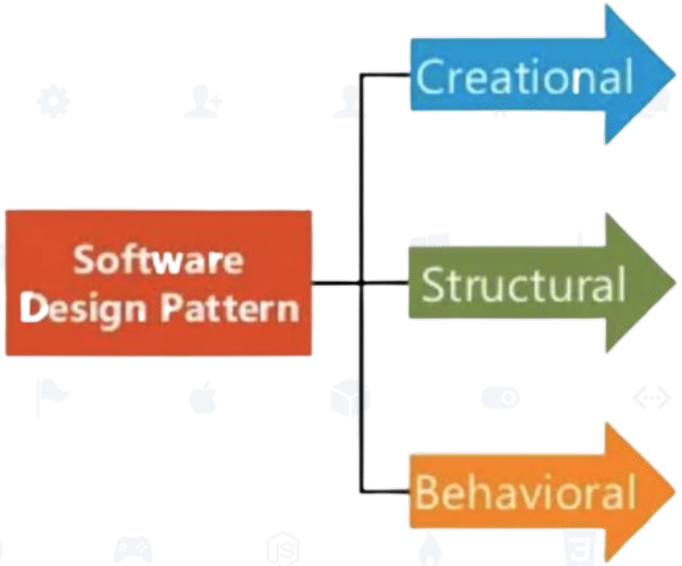
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

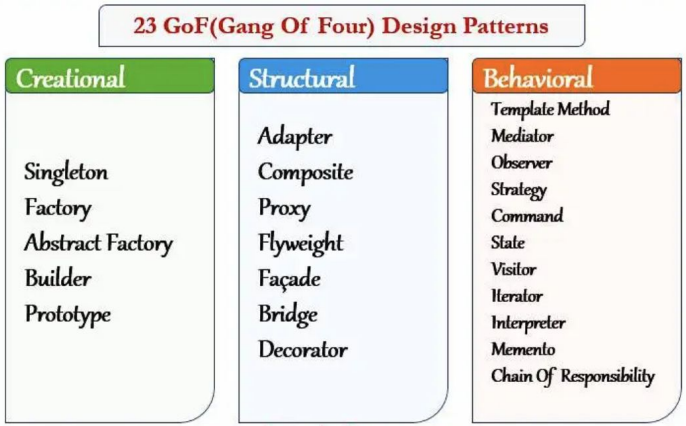
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



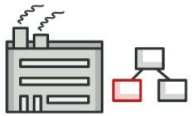
Патерни, що породжують, турбуються про гнучке створення об'єктів без внесення в програму зайвих залежностей.

Структурні патерни показують різні способи побудови зв'язків між об'єктами.

Поведінкові патерни дбають про ефективну комунікацію між об'єктами.

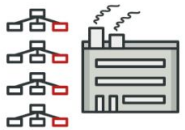


Породжувальні патерни проектування



Фабричний метод
Factory Method

Визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів.

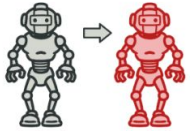


Абстрактна фабрика
Abstract Factory

Дає змогу створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів.

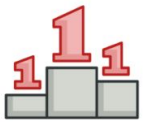
Будівельник
Builder

Дає змогу створювати складні об'єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об'єктів.



Прототип
Prototype

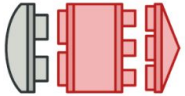
Дає змогу копіювати об'єкти, не вдаючись у подробиці їхньої реалізації.



Одинак
Singleton

Гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

Структурні патерни проектування

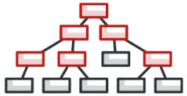


Адаптер
Adapter

Дає змогу об'єктам із несумісними інтерфейсами працювати разом.

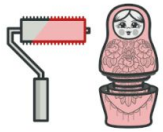
Міст
Bridge

Розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої.



Компонувальник
Composite

Дає змогу згрупувати декілька об'єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об'єкт.

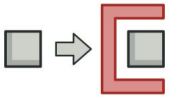


Декоратор
Decorator

Дає змогу динамічно додавати об'єктам нову функціональність, загортаючи їх у корисні «обгортки».

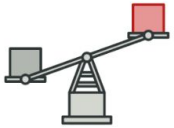
Фасад
Facade

Надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.



Замісник
Proxy

Дає змогу підставляти замість реальних об'єктів спеціальні об'єкти-замінники. Ці об'єкти перехоплюють виклики до оригінального об'єкта, дозволяючи зробити щось *до* чи *після* передачі виклику оригіналові.



Легковаговик
Flyweight

Дає змогу вмістити більшу кількість об'єктів у відведеній оперативній пам'яті. Легковаговик заощаджує пам'ять, розподіляючи спільний стан об'єктів між собою, замість зберігання однакових даних у кожному об'єкті.

Поведінкові патерни проектування

Ланцюжок обов'язків
Chain of Responsibility

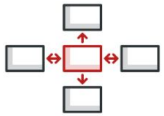
Дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

Команда
Command

Перетворює запити на об'єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.

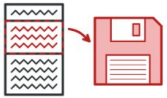
Ітератор
Iterator

Дає змогу послідовно обходити елементи складових об'єктів, не розкриваючи їхньої внутрішньої організації.



Посередник
Mediator

Дає змогу зменшити зв'язаність великої кількості класів між собою, завдяки переміщенню цих зв'язків до одного класу-посередника.

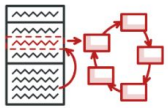


Знімок
Memento

Дає змогу зберігати та відновлювати минулий стан об'єктів, не розкриваючи подробиць їхньої реалізації.

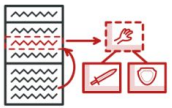
Спостерігач
Observer

Створює механізм підписки, що дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах.



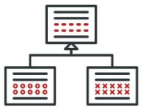
Стан
State

Дає змогу об'єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об'єкта.



Стратегія
Strategy

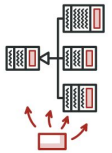
Визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.



Шаблонний метод
Template Method

Визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

Відвідувач
Visitor



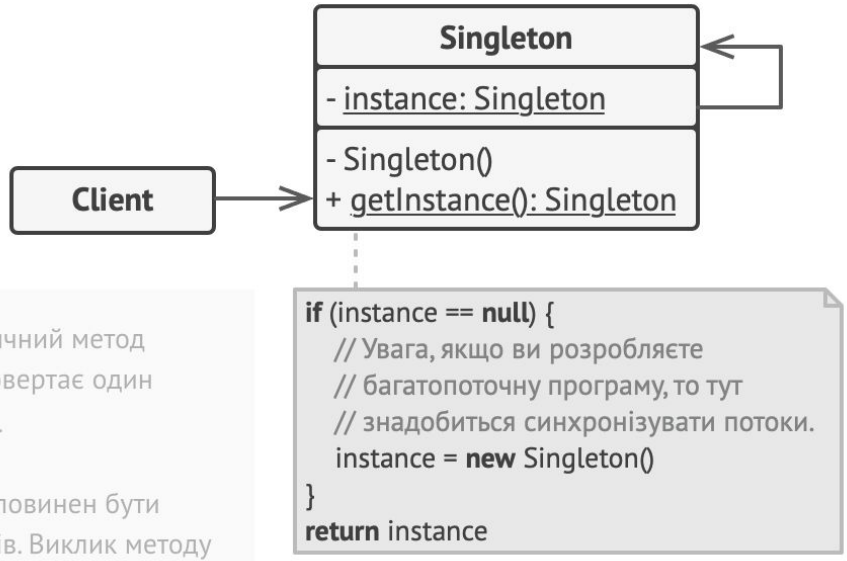
Дає змогу додавати до програми нові операції, не змінюючи класи об'єктів, над якими ці операції можуть виконуватися.

Одинак

Також відомий як: Singleton

Одинак — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

Структура



1 **Одинак** визначає статичний метод `getInstance`, який повертає один екземпляр свого класу.

Конструктор Одинака повинен бути прихований від клієнтів. Виклик методу `getInstance` повинен стати єдиним способом отримати об'єкт цього класу.