Sunday, December 27, 2020 3:31 AM

Single design pattern

Singleton Design pattern - Properties

- Creational design pattern
- Only one instance of the class should exist
- Other classes should be able to get Instance of Singleton class
- Used in Logging, Cache, Session, Drivers

Singleton Design pattern - Implementation

- Constructor should be private
- Public method for returning instance
- Instance type – private static

Initialisation Type:

- Eager Initialisation
- Lazy Initialisation
- Thread safe Method Initialisation
- Thread safe block Initialisation

```
    private Singlepatt() {

    }

    static Singlepatt getinsta() {

        if (single == null) {
            single = new Singlepatt();
            return single;
        } else {
            return single;
        }
    }
}

class Single {
    static Single single;

    private Single() {

    }

    static synchronized Single getinstance() {
        if (single == null) {
            single = new Single();
            return single;
        } else {
            return single;
        }
    }
}
```

Builder Design pattern - Properties

- Creational design pattern
- Used when we have too many arguments to send in Constructor & it's hard to maintain the order.
- When we don't want to send all parameters in Object initialisation

(Generally we send optional parameters as Null)

Used when we have too many arguments to send in constructor & it's hard to maintain order

```
package builder;

class Vehicle {
  //required parameter
  private String engine;
  private int wheel;

  //optional parameter
  private int airbags;

  public String getEngine() {
    return this.engine;
  }

  public int getWheel() {
    return this.wheel;
  }

  public int getAirbags() {
    return this.airbags;
  }

  private Vehicle(VehicleBuilder builder) {
    this.engine = builder.engine;
    this.wheel = builder.wheel;
    this.airbags = builder.airbags;
  }

  public static class VehicleBuilder {
    private String engine;
    private int wheel;

    private int airbags;

    public VehicleBuilder(String engine, int wheel) {
      this.engine = engine;
      this.wheel = wheel;
    }
```

```
ublic static class VehicleBuilder {
  private String engine;
  private int wheel;

  private int airbags;

  public VehicleBuilder(String engine, int wheel) {
    this.engine = engine;
    this.wheel = wheel;
  }

  public VehicleBuilder setAirbags(int airbags) {
    this.airbags = airbags;
    return this;
  }

  public Vehicle build() {
    return new Vehicle(this);
  }


lic class BuilderPatternExample {

ublic static void main(String[] args) {
  Vehicle car = new Vehicle.VehicleBuilder("1500cc", 4).setAirbags(4).build();
  Vehicle bike = new Vehicle.VehicleBuilder("250cc", 2).build();

  System.out.println(car.getEngine());
  System.out.println(car.getWheel());
  System.out.println(car.getAirbags());

  System.out.println(bike.getEngine());
  System.out.println(bike.getWheel());
  System.out.println(bike.getAirbags());
```

ConcurrentHashMap->

- Reads can happen very fast while write is done with a lock.
- You should use ConcurrentHashMap when you need very high concurrency in your project.

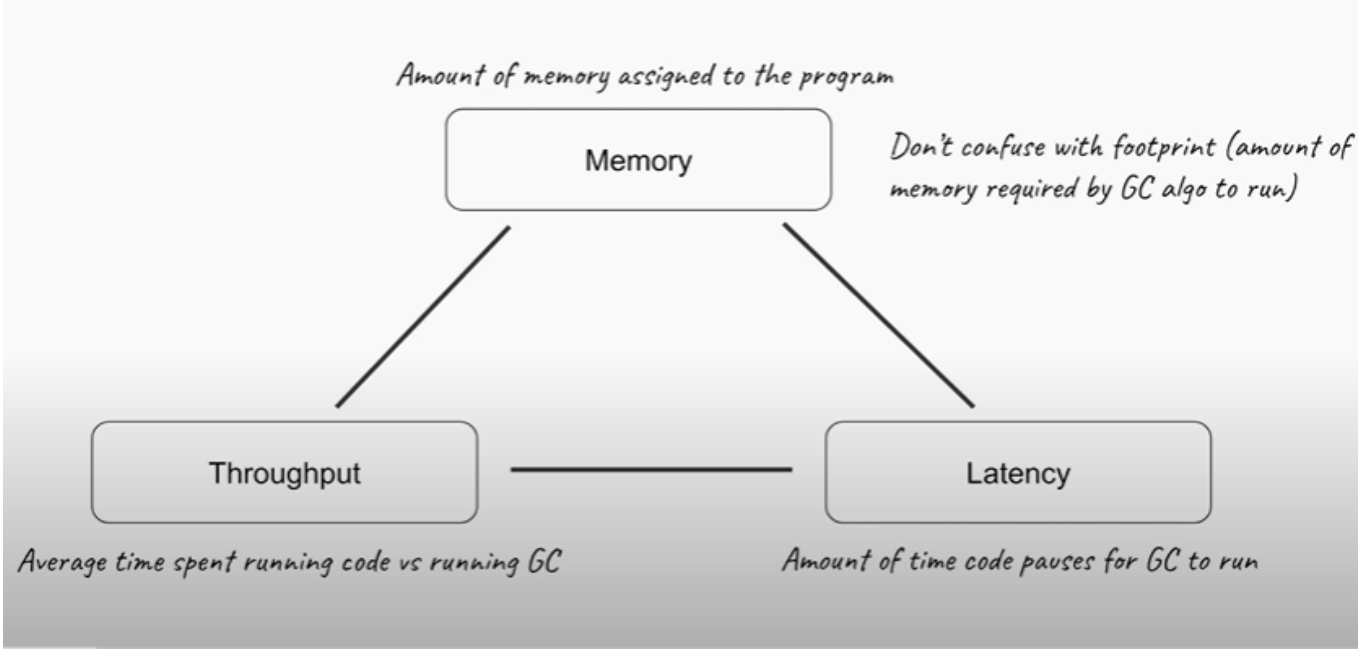
- It is thread safe without synchronizing the whole map.
- Reads can happen very fast while write is done with a lock.
- There is no locking at the object level.

>

SynchronizedHashMap->

- Every read/write operation needs to acquire lock.
- Locking the entire collection is a performance overhead.
- This essentially gives access to only one thread to the entire map & blocks all the other threads.

From <https://crunchify.com/hashmap-vs-concurrenthashmap-vs-synchronizedmap-how-a-hashmap-can-be-synchronized-in-java/>



Garbage collection in java is an automatic process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object.

An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

Main **Advantage** of automatic garbage collection in java is that it removes the burden of manual memory allocation/deallocation from us so that we can focus on problem solving.