Core Java

Table Of Contents

[1. Polymorphism 4](#_Toc521076666)

[2. Abstraction 6](#_Toc521076667)

[3. Encapsulation 7](#_Toc521076668)

[4. Generics 7](#_Toc521076669)

[5. Enumerations 7](#_Toc521076670)

[6. Memory Leak 9](#_Toc521076671)

[6.1 Java Heap Leaks 10](#_Toc521076672)

[7. Equals and Hash code method – Contract between them 23](#_Toc521076673)

[8. Integer vs. int 23](#_Toc521076674)

[9. Composition, Aggregation, Association and Inheritance 24](#_Toc521076675)

[10. Transient 25](#_Toc521076676)

[11. Volatile 25](#_Toc521076677)

[12. Absence of main Method 25](#_Toc521076678)

[13. Interface vs. Abstract Class post Java 8 25](#_Toc521076679)

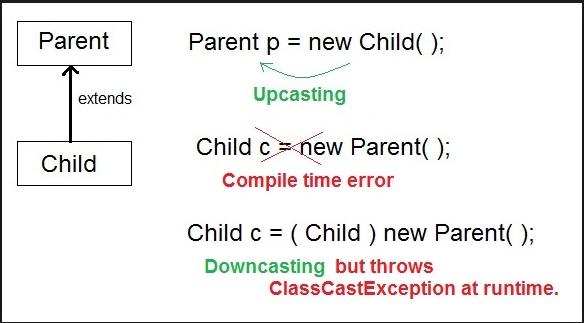
[14. Future Subject 26](#_Toc521076680)

***Core Java***

# Polymorphism

It is the ability of a programming language to process the objects of various class with a single interface. There are basically two types.

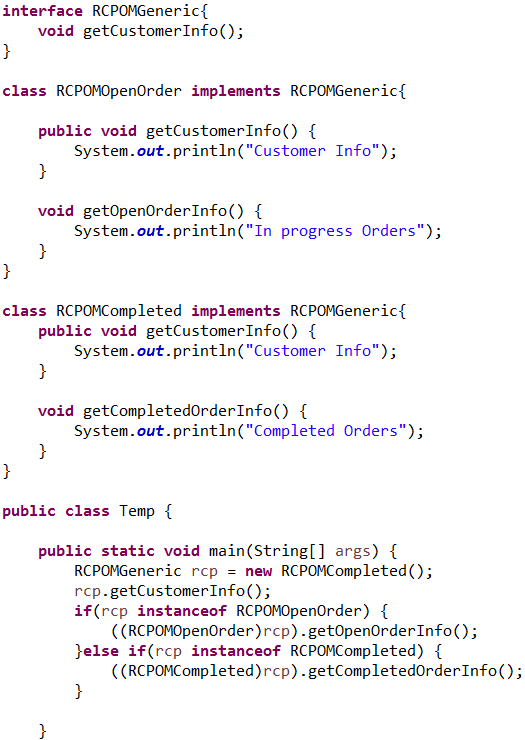
1. *Static Polymorphism*: example – Method Overloading, the mapping of a call to method is associated at compile time.
2. *Dynamic Polymorphism*: example - Method overriding, the association of a call to a method is resolved at run time.
   1. *Up Casting*: The reference of a sub class object is assigned to a super class variable.
   2. *Down Casting*: here the casting is to a sub type. Down in the inheritance tree. But the action would result in compile time error if the assignment is not casted as illustrated below and upon casting the assignment would result in class cast exception at runtime.



Project Example of Down Casting: We have an abstract class having one method which is common among all the extending class and the implementing class can have its own method as per business requirement.

Now during the course of execution first we invoke the common method and later when it is time to invoke the native method of the instance we will not be able to do it through the super class variable and now it would be time for down casting. In order to perform down casting we will have to identify the instance of the variable and based on the instance we down cast the instance and access the native method available in the subtype class.

*Program Snippet for down casting*:



# Abstraction

**It** is the process of used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

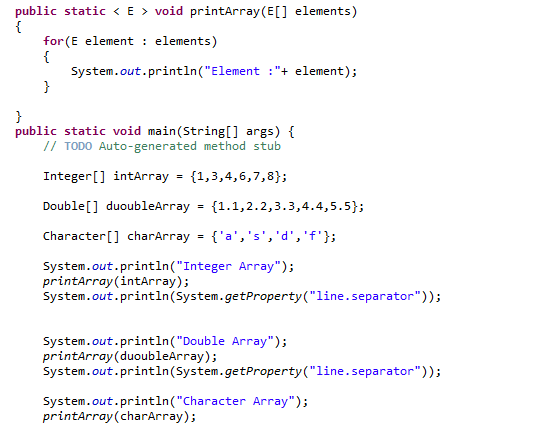
# Encapsulation

**Encapsulation** is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse. One way to think about encapsulation is as a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper.

# Generics

The concept was introduced as part version 5 to allows to extend java’s type system provision a method to process objects of various types while preserving compile time type safety.

*Program*: Generic method to print arrays of different types

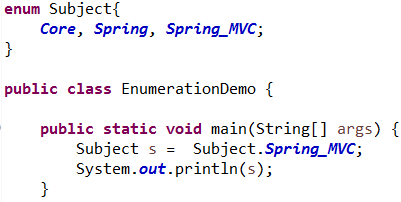


# Enumerations

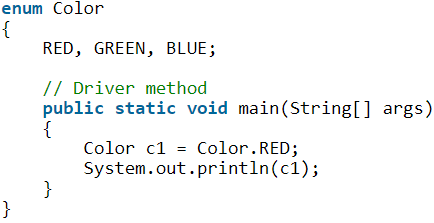
It is used to serve the purpose of representing a group of named constants. For example, suits in a deck of playing cards or days of a week.

Enum is preferred only if we are aware of all the possible values at run time.

Enum Program:



* Enum Constants are public, static and final by default.
* It is also possible to have main method inside enum class which allows us to call the enum directly.



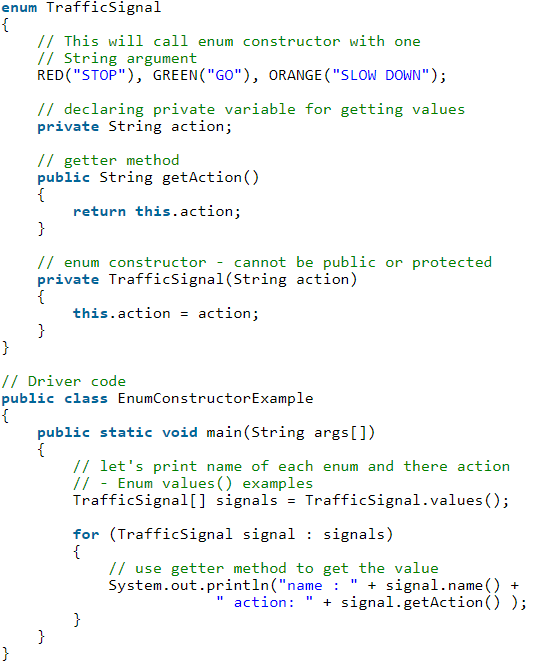
* *Enum and inheritance:*

All the enum internally extend java.lang.enum hence they cannot extend any other class. The toString method is overloaded to print the enum constant.

The prime methods of enum are as follows:

1. Values(): To print all the enum constants availbale.
2. valueOf(): prints the value associated to a enum if available.
3. Ordinal: prints the index of the enum in the given class.

* Enum can have constructor which is executed separately for all the enum constants at the time of enum loading.
* We cannot call the constructor directly.
* It can also have concrete methods.
* ENUM can have customized values. But to do that we have to create variable that can hold the value, a getter method and a parameterized constructor for the same.



# Memory Leak

The standard definition states that when objects are no longer used by the application and garbage collector is no longer able to remove them from the memory because they are still being referenced. As a result, the application consumes more and more resource which eventually leads to one fatal error “OutOfMemoryException”.

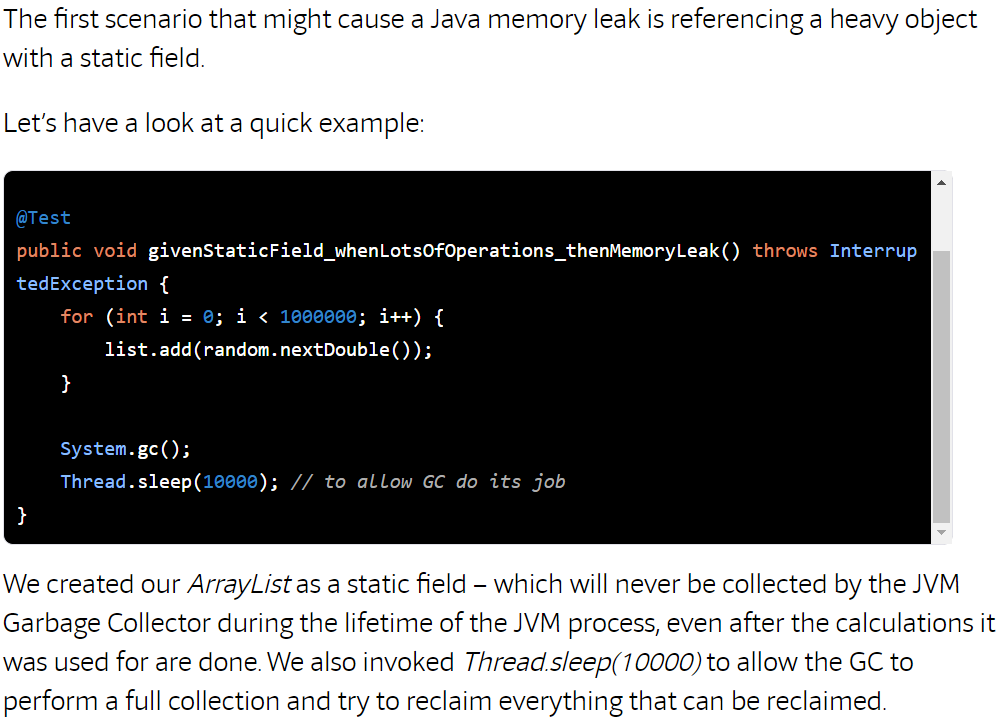
As we can see, we have two types of objects – referenced and unreferenced; the Garbage Collector can remove objects that are unreferenced. Referenced objects won’t be collected, even if they’re actually not longer used by the application.

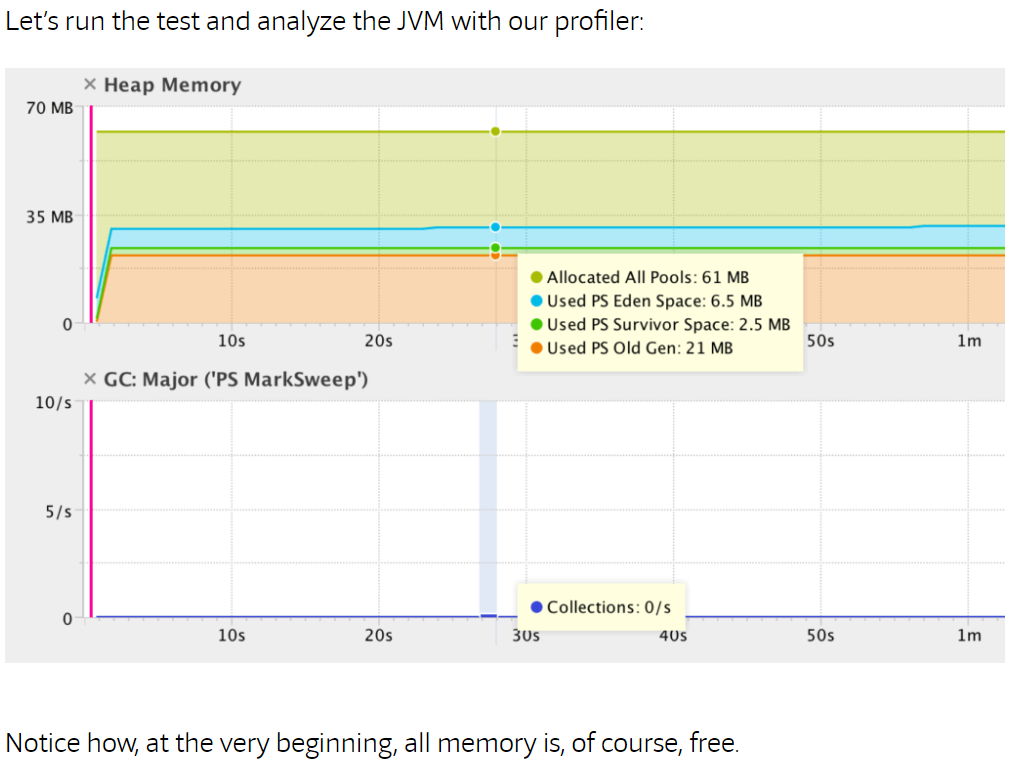
Detecting memory leaks can be difficult. A number of tools perform static analysis to determine potential leaks, but these techniques aren’t perfect because the most important aspect is the actual runtime behavior of the running system.

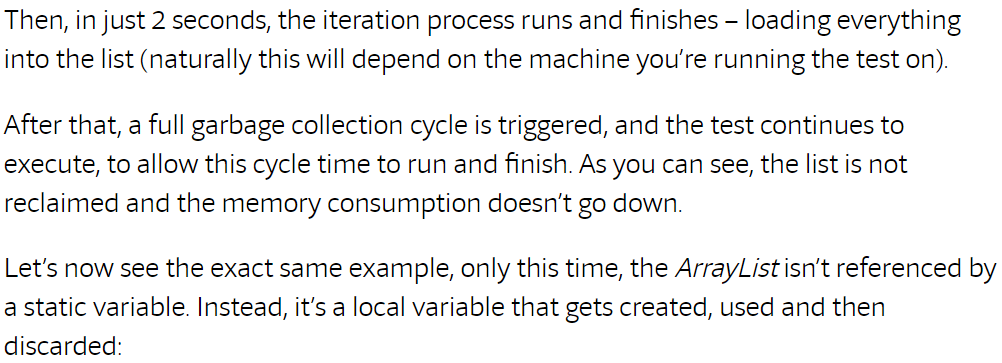
So, let’s have a focused look at some of the standard practices of [preventing memory leaks](https://stackify.com/java-memory-leaks-solutions/), **by analyzing some common scenarios**.

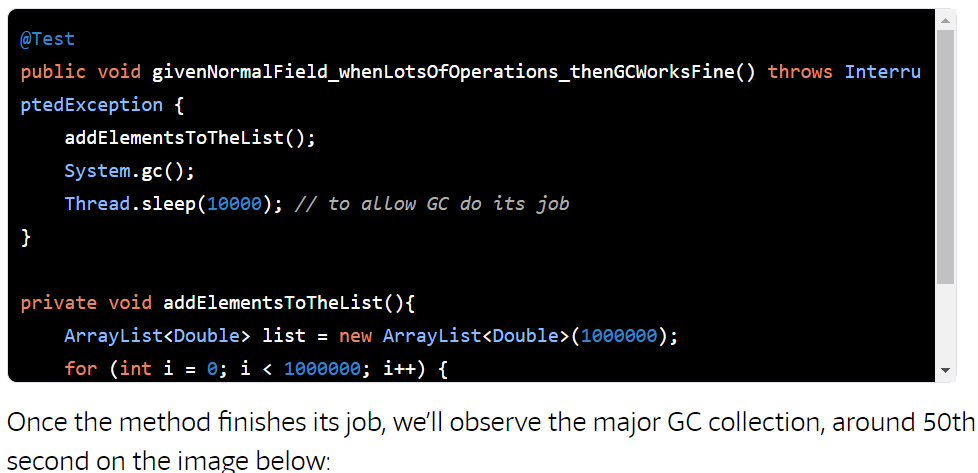
## ****Java Heap Leaks****

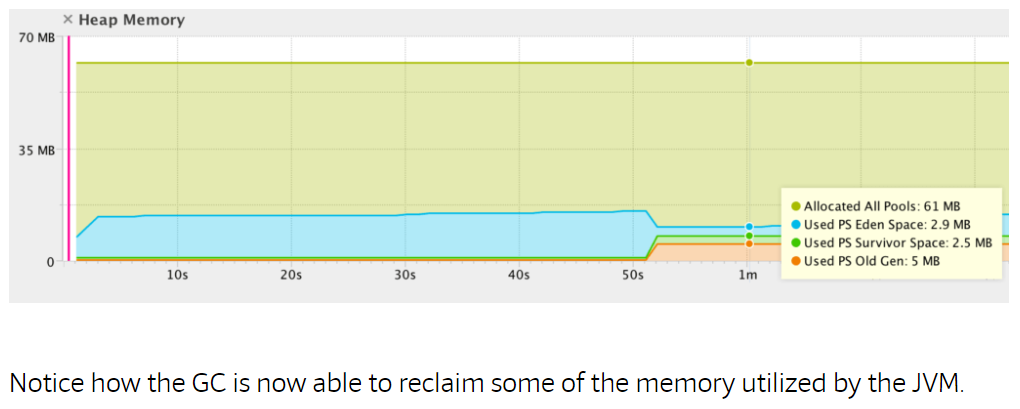
In this initial section, we’re going to focus on the classic memory leak scenario – where Java objects are continuously created without being released.

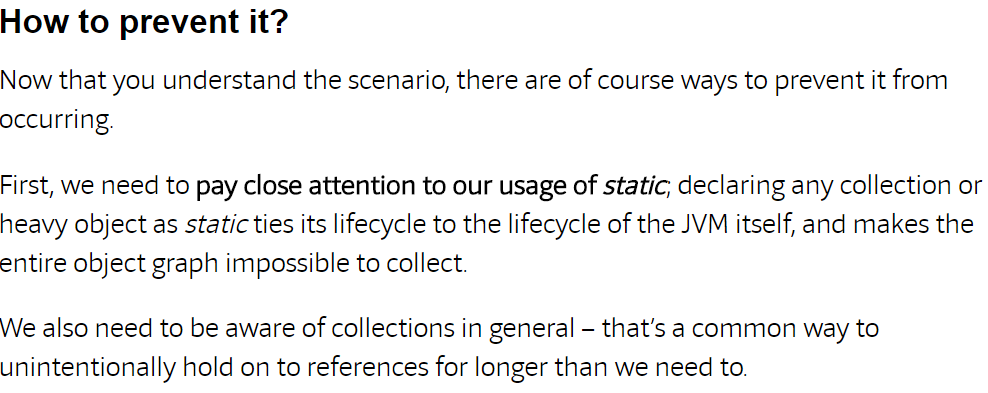


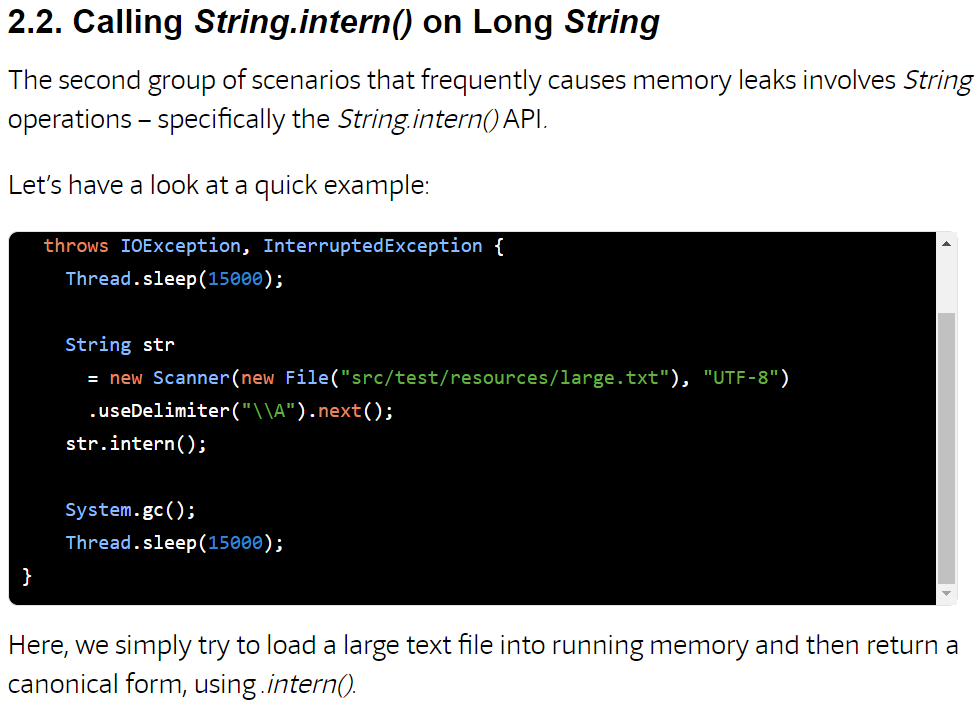


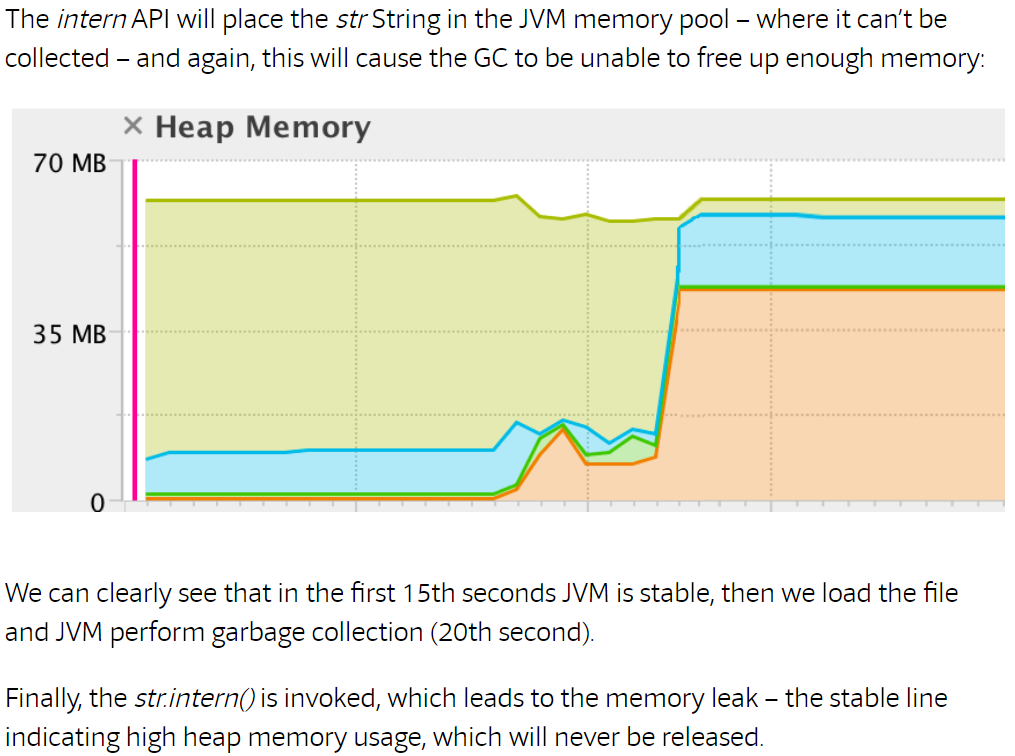


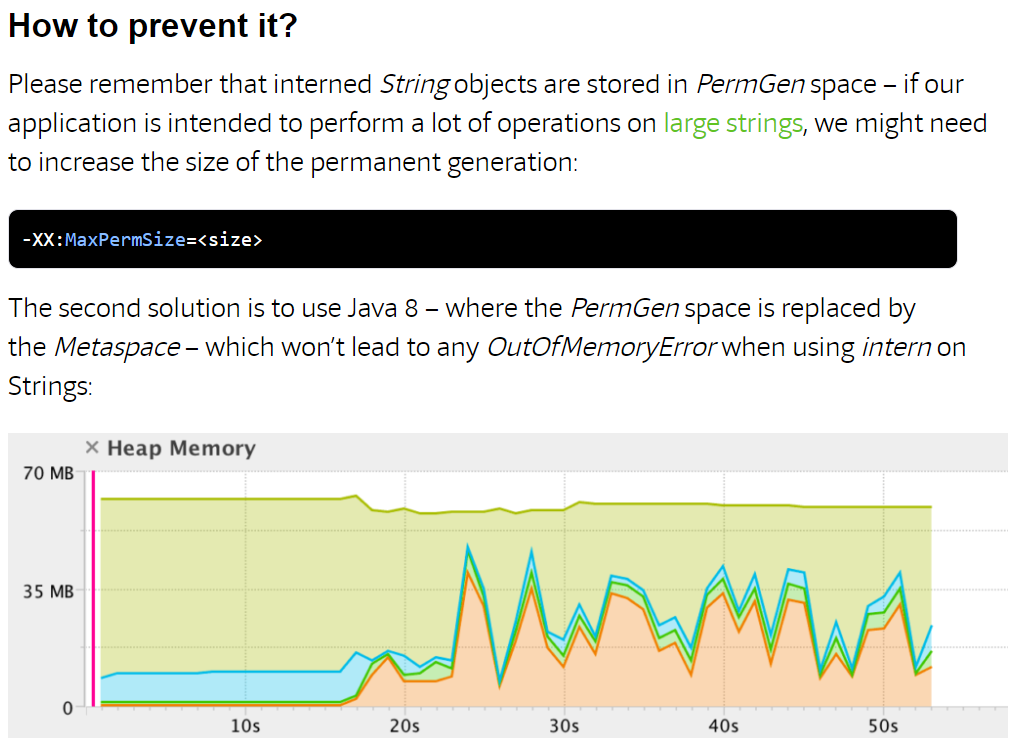


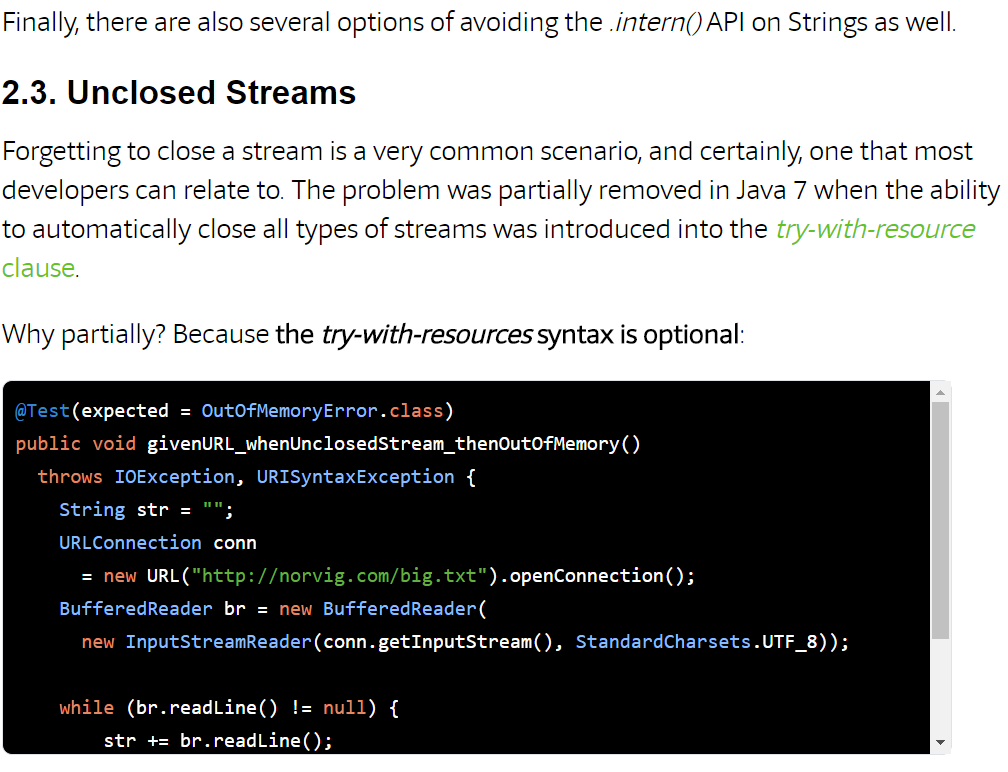


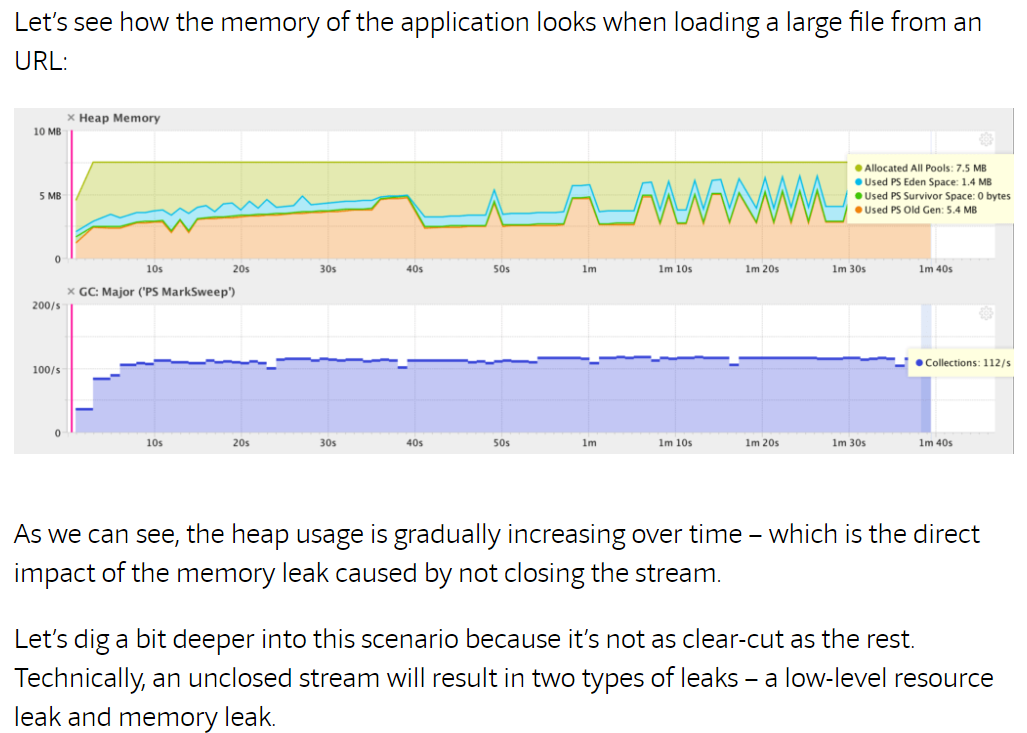


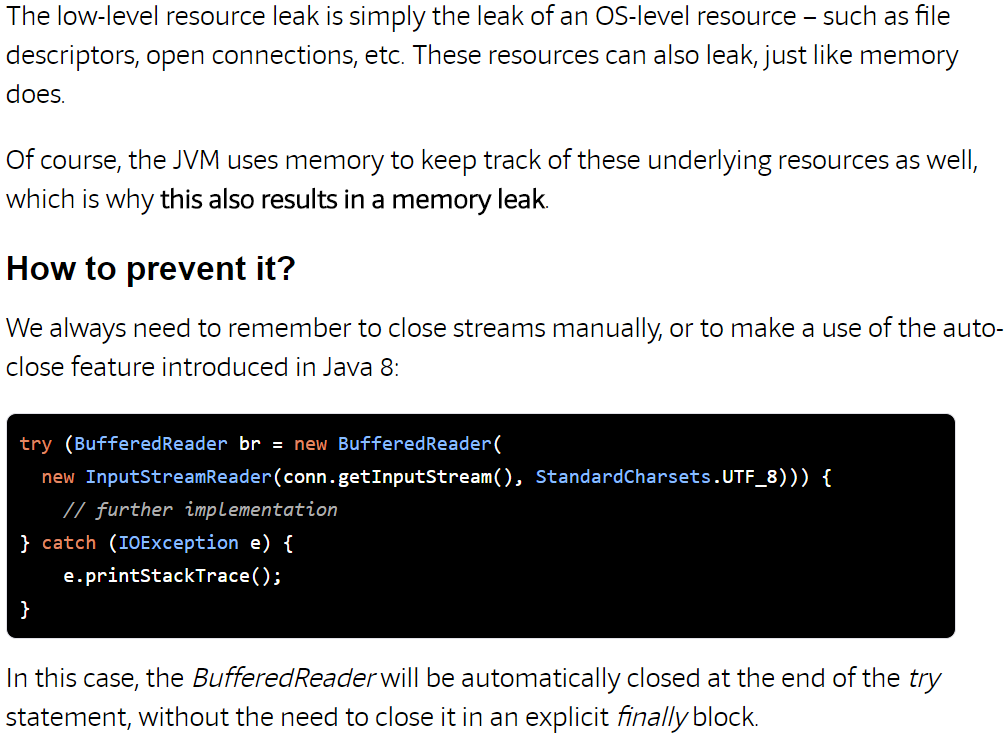


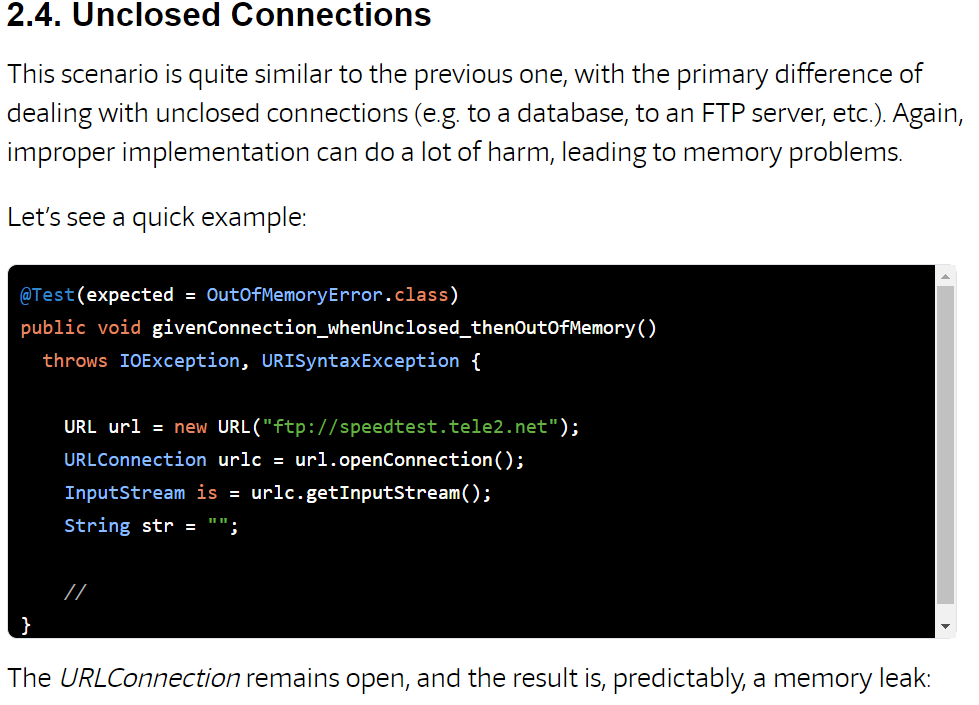


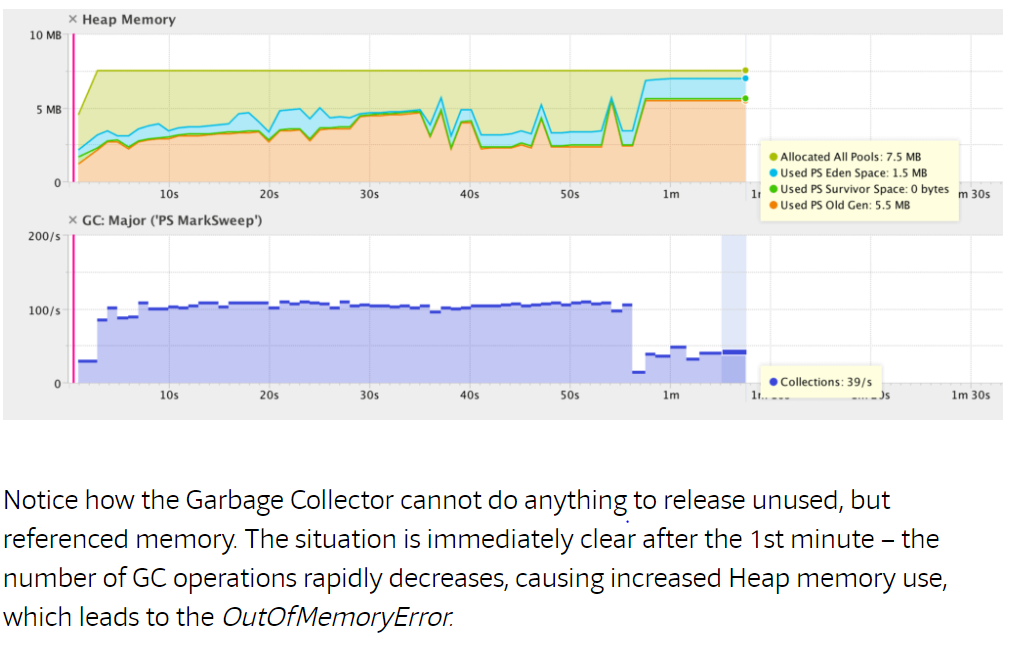


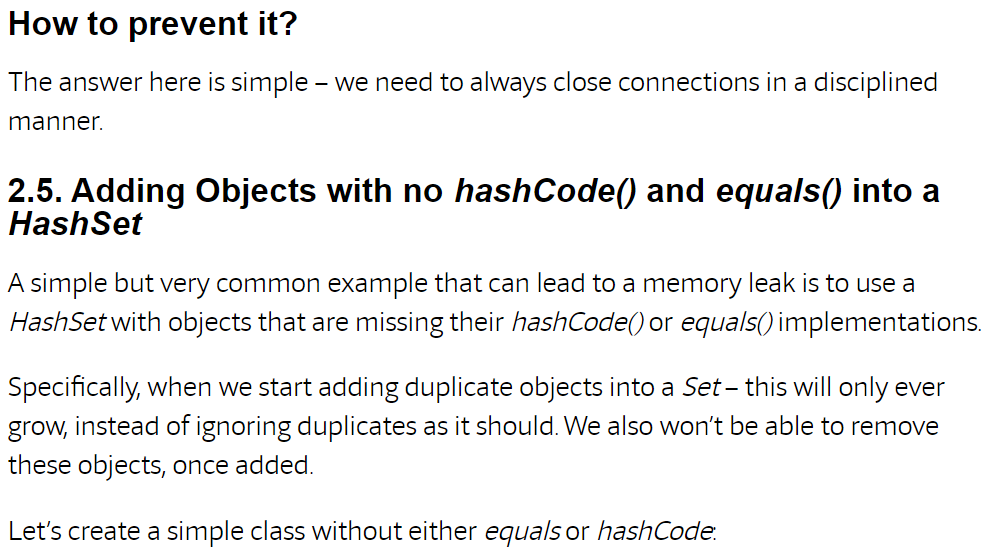


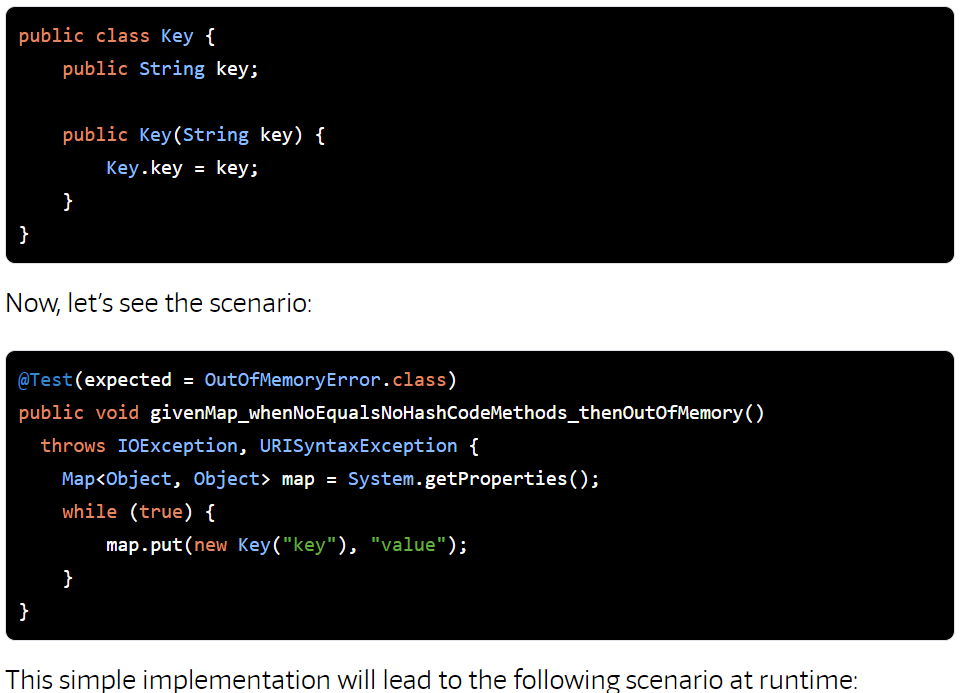


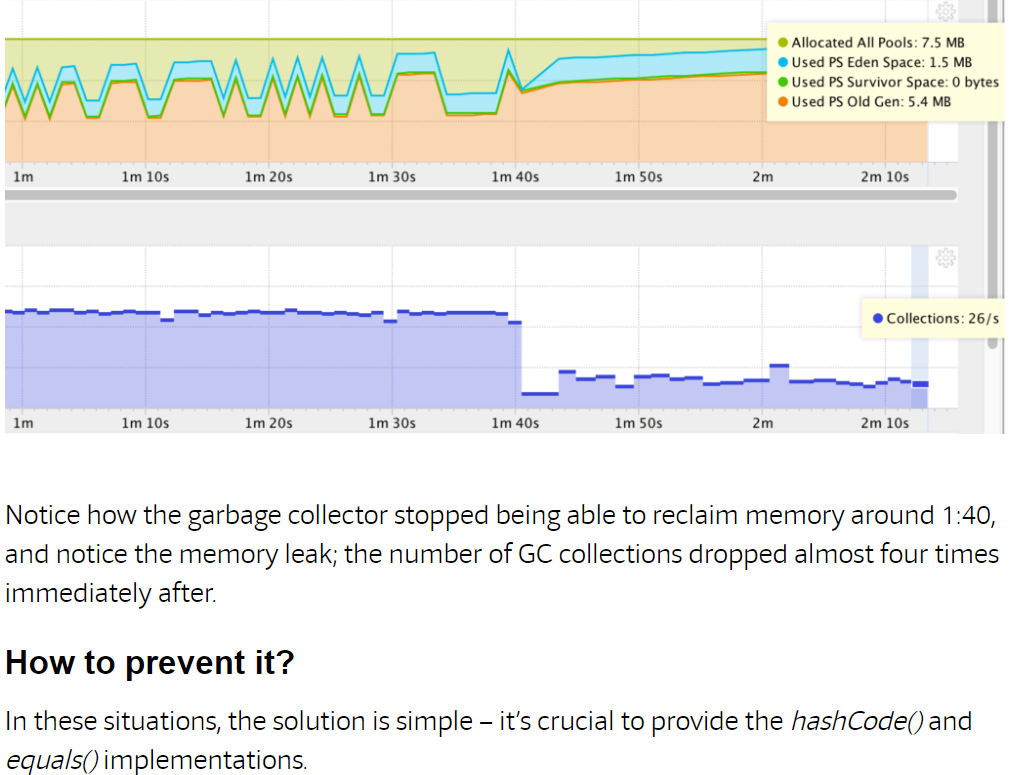


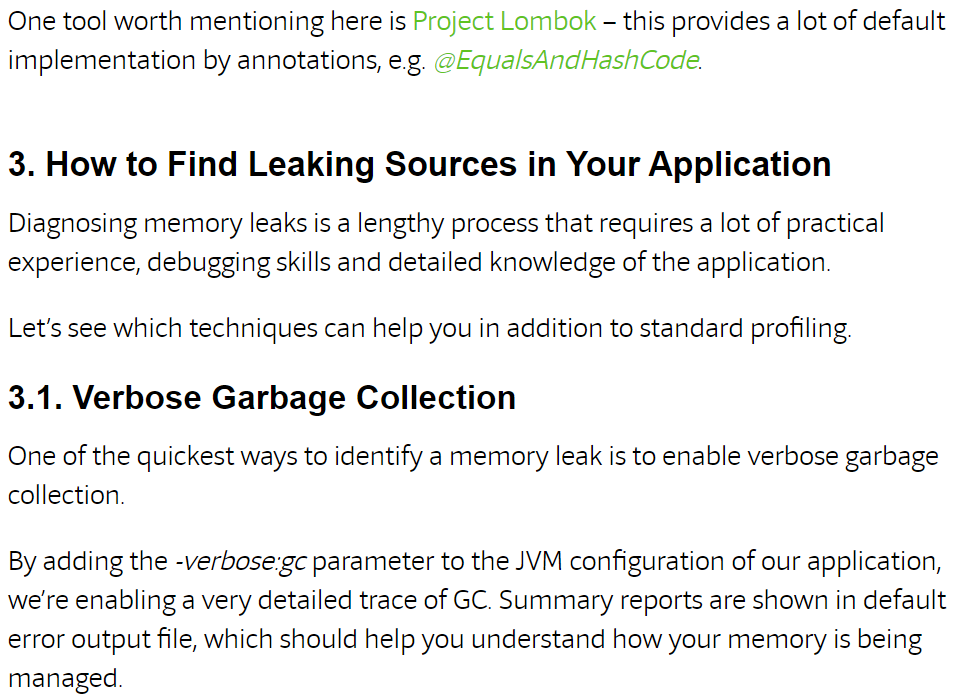


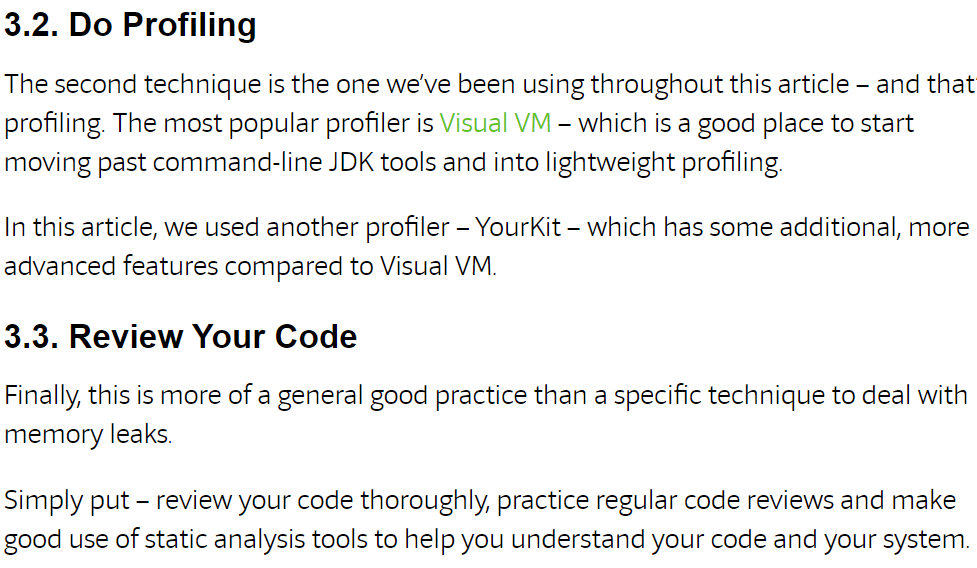








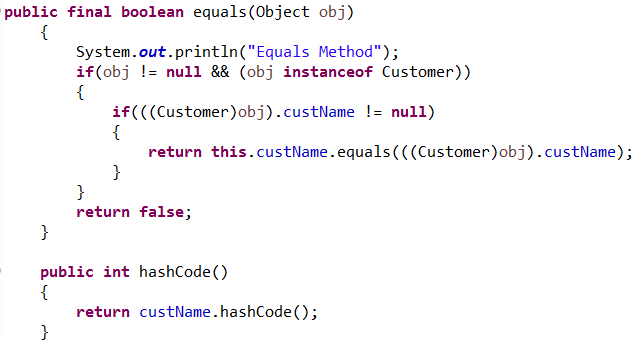




# Equals and Hash code method – Contract between them

The relation plays a vital role in any collection based container where the process of validating the equality of the given two objects is accomplished by overriding the equals and hash code method.

Example:



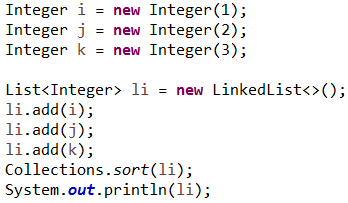
In the case of a primitive data type overriding the equal’s method will do the job, as the hash code method cannot be called against it and also the comparison always takes place in terms of value.

# Integer vs. int

The only difference between int and Integer is that int is a primitive type whereas the Integer is a class which would be used in scenario where in we are to convert a primitive data type to an object type.

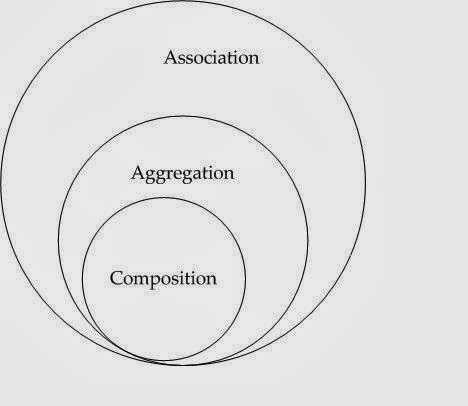
Example Scenario: In the event if we want to sort a sequence of integer value one way is to write our own custom sorting mechanism or to stroe the values as list of Integer values and use the sort method of the Collection interface.

Code Snippet:



# Composition, Aggregation, Association and Inheritance

1. *Association*: It defines the relation between two different classes through their objects. The relation may be one to one, one to many and many to many. Aggregation and Composition are different forms of association.
2. *Composition*: It is a design technique to implement HAS-A relationship between two classes. For example, consider class A and Class B and class A is dependent on Class b and this dependency is achieved by creating an instance of class B within class A. The classes related through composition does not have any meaning with their individual existence.
3. *Aggregation*: It defines a uni-directional relation between two classes (PART-OF relation) and the main difference between composition and aggregation is that the classes that are related has a business value with their individual existence.
4. *Inheritance*: The only difference between composition and inheritance the dependency between the classes is achieved by means of inheritance rather than creating an instance of the dependent class.



# Transient

The keyword indicates that the member is to be excluded while serializing the object.

# Volatile

Declaring a variable as volatile makes the value to be never cached that is any modification goes directly onto the main memory access to this variable acts as through it is enclosed with a synchronized block. In short in a multi-threaded environment the volatile keyword can be used on a variable to have the latest value of the computation.

# Absence of main Method

In the absence of main method we can still execute a java program but it depends on the available execution model.

Say if we have a servlet or applet which is also a java program running on a browser does not need any main method as the execution of it is taken care by the lifecycle methods like init, service and destroy.

But when the question is just executing a core java program the answer is no, we cannot execute a program without main function. There might be a discussion if we could do it through static block which is partially correct as static block gets executed prior to main method but then again, we can execute anything through static block.

# Interface vs. Abstract Class post Java 8

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***Interface*** | ***Abstraction*** |
| Variables | Can have only static and final variables | Supports final, non-final variables, static, non-static |
| Implementation | Interface cannot provide implementation of abstract class | Can provide implementation of interface |
| Multiple Implementation | An interface can extend another interface only | an abstract class can extend another java class and implement an interface |
| Accessibility of members | Members are public by default | Can have private and protected members |

|  |  |  |
| --- | --- | --- |
| ***SI NO*** | ***Topic*** | ***Status*** |
| 1 | Writing Custom Hash Map |  |
| 2 | Custom String class which creates objects in the same way as that of String constant pool(immutable) |  |
| 3 | Enhance hashmap to store values in ascending or descending order. |  |
| 4 | Enhance hashmap to store keys in ascending or descending order. |  |
| 5. | Memory modem of latest java version. |  |
| 6. | Custom Hash Map, Custom Iterator |  |
| 7. | OutOfMemory Exception : What, How to analyse and how to resolve |  |
| 8. | Concept of Transaction and session Management |  |
| 9. | How to implement Data source. |  |
|  |  |  |
|  |  |  |

# Future Subject

|  |  |  |
| --- | --- | --- |
| ***SI NO*** | ***Topic*** | ***Status*** |
| 1 | Basic Restful Webservice, Advanced Restful webservice |  |
| 2 | Maven |  |
| 3 | Junit |  |
| 4 | Spring Security |  |
| 5. | Java Script |  |
| 6. | Database Concepts |  |
| 7. | Class Designing Tool |  |
| 8. |  |  |
| 9. |  |  |
|  |  |  |
|  |  |  |