**DAY 10 TASKS**

**Learn best programming practices for Java**

## Prefer returning Empty Collections instead of Null

If a program is returning a collection which does not have any value, make sure an Empty collection is returned rather than Null elements. This saves a lot of “if else” testing on Null Elements.

## Use Strings carefully

If two Strings are concatenated using “+” operator in a “for” loop, then it creates a new String Object, every time. This causes wastage of memory and increases performance time. Also, while instantiating a String Object, constructors should be avoided and instantiation should happen directly.

## Avoid unnecessary Objects

One of the most expensive operations (in terms of Memory Utilization) in Java is Object Creation. Thus it is recommended that Objects should only be created or initialized if necessary.

## Dilemma between Array and ArrayList

Developers often find it difficult to decide if they should go for Array type data structure of ArrayList type. They both have their strengths and weaknesses. The choice really depends on the requirements.

* Arrays have fixed size but ArrayLists have variable sizes. Since the size of Array is fixed, the memory gets allocated at the time of declaration of Array type variable. Hence, Arrays are very fast. On the other hand, if we are not aware of the size of the data, then ArrayList is more data will lead to ArrayOutOfBoundException and less data will cause wastage of storage space.
* It is much easier to Add or Remove elements from ArrayList than Array
* Array can be multi-dimensional but ArrayList can be only one dimension.

## When Finally does not get executed with Try

Consider following code snippet:

|  |  |
| --- | --- |
|  | public static void main(String[] args) { |

|  |  |
| --- | --- |
|  | for(int i=0;i<5;i++) |

|  |  |
| --- | --- |
|  | { |

|  |  |
| --- | --- |
|  | try { |

|  |  |
| --- | --- |
|  | if(i==4) { |

|  |  |
| --- | --- |
|  | System.out.println ("Inside Try Block.Exiting without executing Finally block."); |

|  |  |
| --- | --- |
|  | System.exit(0); |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | finally { |

|  |  |
| --- | --- |
|  | System.out.println ("Inside Finally Block."); |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | }  From the program, it looks like “println” inside *finally*block will be executed 5 times. But if the program is executed, the user will find that *finally* block is called only 4 times. In the fifth iteration, *exit* function is called and *finally* never gets called the fifth time. The reason is- System.exit halts execution of all the running threads including the current one. Even *finally* block does not get executed after *try* when *exit* is executed.  When *System.exit* is called, JVM performs two cleanup tasks before shut down:  First, it executes all the *shutdown hooks*which have been registered with *Runtime.addShutdownHook*. This is very useful because it releases the resources external to JVM.  Second is related to *Finalizers*. Either *System.runFinalizersOnExit* or *Runtime.runFinalizersOnExit*. The use of *finalizers*has been deprecated from a long time. *Finalizers*can run on live objects while they are being manipulated by other threads. This results in undesirable results or even in a deadlock. |

## Check Oddity

public boolean oddOrNot(int num) {

return num % 2 == 1;

}

These lines seem correct but they will return incorrect results one of every four times (Statistically speaking). Consider a negative Odd number; the remainder of division with 2 will not be 1. So, the returned result will be false which is incorrect!

This can be fixed as follows:

public boolean oddOrNot(int num) {

 return (num & 1) != 0;

}

Using this code, not only is the problem of negative odd numbers solved, but this code is also highly optimized.

## Difference between single quotes and double quotes

public class Haha{

public static void amin(String args[]){

System.out.println(“H”+”a”);

System.out.println(‘H’+’a’);

}

}

|  |  |
| --- | --- |
|  | From the code, it would seem return “HaHa” is returned, but it actually returns Ha169. The reason is that if double quotes are used, the characters are treated as a string but in case of single quotes, the char -valued operands ( ‘H’ and ‘a’ ) to int values through a process known as widening primitive conversion. |

## Avoiding Memory leaks by simple tricks

Memory leaks often cause performance degradation of software. Since, Java manages memory automatically; the developers do not have much control. But there are still some standard practices which can be used to protect from memory leakages.

* Always release database connections when querying is complete.
* Try to use Finally block as often possible.
* Release instances stored in Static Tables.

1. **Avoiding Deadlocks in Java**

Deadlocks can occur for many different reasons. There is no single recipe to avoid deadlocks. Normally deadlocks occur when one synchronized object is waiting for lock on resources locked by another synchronized object.

## Reserve memory for Java

Some of the Java applications can be highly CPU intensive as well as they need a lot of RAM. Such applications generally run slow because of a high RAM requirement. In order to improve performance of such applications, RAM is reserved for Java. So, for example, if we have a Tomcat webserver and it has 10 GB of RAM. If we like, we can allocate RAM for Java on this machine using the following command:

export JAVA\_OPTS="$JAVA\_OPTS -Xms5000m -Xmx6000m -XX:PermSize=1024m -XX:MaxPermSize=2048m”

* Xms = Minimum memory allocation pool
* Xmx = Maximum memory allocation pool
* XX:PermSize = Initial size that will be allocated during startup of the JVM
* XX:MaxPermSize = Maximum size that can be allocated during startup of the JVM

## How to time operations in Java

There are two standard ways to time operations in Java: **System.currentTimeMillis()**and **System.nanoTime()**

The question is which of these to choose and under what circumstances. In principle, they both perform the same action but are different in the following ways:

* System.currentTimeMillis takes somewhere between 1/1000th of a second to 15/1000th of a second (depending on the system) but System.nanoTime() takes around 1/1000,000th of a second (1,000 nanos)
* System.currentTimeMillis takes a few clock cycles to perform Read Operation. On the other hand, System.nanoTime() takes 100+ clock cycles.
* System.currentTimeMillis reflects Absolute Time (Number of millis since 1 Jan 1970 00:00 (Epoch Time)) but System.nanoTime() does not necessarily represent any reference point.

## Choice between Float and Double

Double is often preferred over float in software where precision is important because of the following reasons:

Most processors take nearly the same amount of processing time to perform operations on Float and Double. Double offers far more precision in the same amount of computation time.

## Computation of power

To compute power (^), java performs Exclusive OR (XOR). In order to compute power, Java offers two options:

* **Multiplication**
* **pow(double base, double exponent):** ‘pow’ method is used to calculate where multiplication is not possible (base^exponent)

## How to handle Null Pointer Exceptions

It’s a good idea to check Nulls early so that they can be eliminated.

## Simple String Search

Java offers a Library method called indexOf(). This method is used with String Object and it returns the position of index of desired string. If the string is not found then -1 is returned.

## A Simple IO

In order to read from a file and write to a file, Java offers FileInputStream and FileOutputStream Classes. FileInputStream’s constructor accepts filepath of Input File as argument and creates File Input Stream. Similarly, FileOutputStream’s constructor accepts filepath of Output File as argument and creates File Output Stream.After the file handling is done, it’s important to “close” the streams.

## Simple Java Swing Example

With the help of Java Swing GUI can be created. Java offers Javax which contains “swing”. The GUI using swing begin with extending JFrame. Boxes are added so they can contain GUI components like Button, Radio Button, Text box, etc. These boxes are set on top of Container.

## Measuring time

Many applications require a very precise time measurement. For this purpose, Java provides static methods in System class:

* **currentTimeMillis():** Returns current time in MilliSeconds since Epoch Time, in Long.
* **nanoTime():** Returns the current value of the most precise available system timer, in nanoseconds, in long. nanoTime() is meant for measuring relative time interval instead of providing absolute timing.

#### Use Collections

Java is shipped with a few collection classes – for example, Vector, Stack, Hashtable and Array. The developers are encouraged to use collections as extensively as possible for the following reasons:

* + Use of collections makes the code reusable and interoperable.
  + Collections make the code more structured, easier to understand and maintainable.
  + Out of the box collection classes are well tested so the quality of code is good.

#### 10-50-500 Rule

In big software packages, maintaining code becomes very challenging. Developers, who join fresh ongoing support projects, often complain about: Monolithic Code, Spaghetti Code. There is a very simple rule to avoid that or keep the code clean and maintainable: 10-50-500.

* + 10: No package can have more than 10 classes.
  + 50: No method can have more than 50 lines of code.
  + 500: No class can have more than 500 lines of code.

#### Usage of Design Patterns

Design patterns help developers to incorporate best Software Design Principles in their software. They also provide common platform for developers across the globe. They provide standard terminology which makes developers to collaborate and easier to communicate to each other.

#### Document ideas

Never just start writing code. Strategize, Prepare, Document, Review and Implementation. First of all, jot down your requirements. Prepare a design document. Mention assumptions properly. Get the documents peer reviewed and take a sign off on them.

#### Use Equals over ==

== Compares object references, it checks to see if the two operands point to the same object (not equivalent objects, the same object).On the other hand, “equals” performs actual comparison of two strings.

#### Avoid Floating Point Numbers

Floating point numbers should be used only if they are absolutely necessary. For example, representing Rupees and Paisa using Floating Point numbers can be Problematic – BigDecimal should instead be preferred. Floating point numbers are more useful in measurements.