Threads ThreadWaitExample

*Multi Tasking*

*Multi tasking is an operation in which multiple tasks are performed simultaneously. Multitasking is used to utilize CPU ideal time.*

**Multi tasking are of two types**

1) Process based multitasking ( Multi Processes)

2) Thread based Multitasking ( Multi Threading)

**Process based Multitasking**

*In Process-based multi tasking or multi processes are executed Simultaneously. You are all familiar with process-based multi tasking. Because of this feature only, your computer runs two or more program Simultaneously.*

*For example : you can play video and print a text.*

**Thread based Multitasking or Multi-threading**

In thread-based multi-tasking or multi-threading, multiple threads in a process are executed simultaneously (at same time).

For example: Ms word print a document at the same time thread can accept user inputs.

**Processes:** Process is an executing instance of an application.

**Threads:** thread is smallest executable unit of a process.

Two ways a thread can be created

1) java.lang.Thread class

We can create a thread by extending thread class and override run method.

2) java.lang.Runnable interface. Is having only one method I.e run()

Class MyThread implements Runnable

{

@Override

Public void run(){

///business logic

}

}

After defining a thread, create an object to java.lang.Thread class, through a constructor which takes Runnable type as an argument

MyThread thread = new MyThread();

thread.start();

Four ways we can define/create a thread.

Thread as a separate concrete class.

Thread as a Nested class or Static inner class

Thread as a Member Inner class.

Thread as a Anonymous Inner class.

Use class it self as a thread.

Thread as a local Inner class.

Types of threads in

User defined thread

Daemon thread.

**UserDefined Thead**

User defined threads are threads which are created by the application or user.

These are high priority threads.

These threads are foreground threads.

**Daemon Thread**

Daemon thread are threads which are created by JVM

These threads always run in background tasks like Garbage collection and House keeping tasks.

These are low priority threads.

JVM doesn’t wait for daemon threads to finish their task.

User threads can be converted to daemon threads.

Example : userThread.setDemon(true);

We can’t set thread as daemon thread. Once thread started, it will throw Illigal threadStateException.

A thread can be checked whether it is Daemon thread or not by using following method.

UserThread.isDaemon()

-> true

-> false

Naming a Thread in java

Pubic final void setName(String name);

Public final String getName()

We can change the name of a thread at any stage/state of the thread.

SetName() method may throw a SecurityException at runtime if the current thread can’t modify the name of specified thread.

**Default Name of a Thread.**

All threads have names, thread will get default name.

The name of the thread follow two words. “Thread” followed by hyphen (-) and followed by an integer number starting with 0

getName() method we can fetch thread name.

Another way of creating thread.

MyThread t = new MyThread(“MyThread”);

**How to Identify a thread**

By using getId() method of thread class

To get main thread Thread t = Thread.currentThread();

t.getId();

We can’t assign our won id to a thread.

**Priority of a thread in java**

Based on priority threads will be executed. Java.lang.Thread.

SetPriority(int value);

GetPriority();

There are 3 Constans fields in java.lang.Thread class related to priorities of a thread.

MIN\_PRIORITY its value 1

NORM\_PRIORITY its value 5

MAX\_PRIORITY its value 10

SetPriority() method may throw IllegalArgument Exception if supplied priorities MIN\_PRIORITY and MAX\_PRIORITY.

SecurityException() if current thread can’t modify the priorities of a specified thread.

We can change Priorate of thread using set Priorate method.

Thread t = Thread.currentThread();

t.setPriorite(5);

**Thread.sleep()**

Makes the currently executing thread to pause its execution for a specified period of time.

Two overloaded forms of sleep method.

1) public static void sleep(long millis) throws InterruptException.

2) public static void sleep(long millSec, long nanoSec)

Throws Interrupt Exception.

Thread.sleep() method throws interruptException, If a thread is interrupted by other thread.

InterruptedException is a checked exception, it should be in try catch block or it must be spedified with throws clauses.

Thread.sleep(5000) - 5sec

1. It is always current thread that is going to sleep. For example in the below example, main thread is going to sleep not userThead. Even though we called sleep() method on MyUserThread.

Class MyUserThread extends Thread {

public run() {

}

}

Public class ThreadsJave {

public static void main(String args[]) {

MyUserThread t = new MyUserThead();

t.start();

try{

t.sleep();

} catch(Exception e) {

S.o.p(e);

}

}

}

IT’s bad pratic to call sleep method on thread class. If we want paticular thread to sleep for a while then call sleep() method inside the run() of the thread.

Thread.sleep() method may also throw illigalArgument Excption if millis value is negative

It should not in the range of [0 - 99999]

*When a thread is going to sleep, it does not release the synchronized locks it holds*.

**Join():** method of java.lang.Thread class is used to maintain the order of the execution of threads.

Using join() method, we can make the currently executing thread to wait for the some other threads finish the task.

Let’s us assume that there are two threads namely thread1 and thread2 , you can make thread1 to hold its’s execution for sometime, So that thread2 can finish the task. For this to happen we should call join() method on thread2 with in Thread1.

Join() is also overloaded. There are three forms of join() method available.

Public final void join() throws InterruptedException.

join(mills)

join(mills, nanos)

Join() method also throws interrupted exception, so we need to keep join() in try catch block.

Example :

**Synchronization :**

Is a strategy or a method to avoid thread interference and hence protecting data from inconsistency

Synchroniation is also one of the ways to make code thread safe.

Synchroniation in java is implemented using Synchronized keyword.

Synchronized keyword can be used with methods or blocks but not with variables.

When method or block declared as synchronised only one thread can enter into that method or block.

When one thread is executing synchronised method or block, the other threads which wants to execute tha`t method or block, wait or suspend their execution until first thread is done with the method or block. Thus avoiding the thread interference and achieving safeness. This can be explained well with the help of an example.

Example 1 : Synchronization1

**The Logic Behind Synchronization in java**

The Synchronization in java build around an entity called object lock or monitor, Here is brief description about lock or monitor.

When ever an object is created to any class, an object lock is created and stored inside the object.

One object will have only one object lock associated with it.

Any threads wants to entre into synchronized methos or blocks of any objects, they must acquire object lock associated with that object and release the lock after they have done with the execution.

The other thread which want to enter into synchronized method of that objects have to wait until the currently executing thread release the object lock.

To enter into static synchronized method or blocks thread have to acquire class lock associated with that class as static members are stored inside the class memory.

**Synchronized Blocks:**

Sometime you need only some part of the method to be Synchronised not the whole method

This can be achieved by Synchronized blocks.

Synchronized block must be defined inside a definition blocks like methods, Constructors, static initializer or instance initializer.

***Synchronized blocks takes one argument called mutex.***

If synchronized block is defined inside non-static definition blocks like non-static methods. Instance initializer or constructor, then this mutex must be an instance of that class.

If synchronized block is defined inside static definition blocks like static methods or static initializer, then this mutext must be like className class.

Points to remember in Synchronization block

1. You can use Synchronized keyword only with methods but not with variables constructs, static initializer and instance initializers.

will get compile time error.

1. Constructor, static initializer and instance inilizer can not be declared with synchronized keyword. But can contain synchronized blocks.

3) Both static and non static methods can use synchronized keyword.

4) It is possible that both static synchronized and not static Synchronized methods can run simultaneously , because static methods need class level lock and Non static methods need object level lock.

5) A method can contain any number of Synchronized blocks. This is like synchronizating multiple parts of a method.

6) Synchronization blocks can be nested.

7) Lock acquired by the thread before executing Synchronized method or block must be released after the compilation of execution. No matter where execution is completed normally or abnormally (due to exceptions).

8) Synchronzation in java Re-entrant in nature, A thread cannot acquire a lock that is owned by another thread. But a thread can acquire a lock that is already owns. That means if a synchronized method gives a call to another synchronized method which need same lock, then currently executing thread can directly enter into that method or block without acquiring the lock.

9) synchronized method or block is very slow. They decrease the performance of an application. So special care need to be taken While using synchronization. Use synchronize only when you need it the most.

10) Use Synchronized blocks instead of synchronized methods. Because synchronizing some part of a method improves the performance than synchronizing the whole method.

**DeadLock**

Dead lock in java is a condition which occurs two or more threads get blocked waiting for each other for an infinite period of time to release the resource lock they hold.

Example : DeadLockExample.java

**How to detect DeadLock**

Programmatically you can detect the threads which have entered into deadlock condition and also retrieve the details of them using ThreadMxBean interface of java.lang.Management package.

1. We need to get **ThreadMxBean** using getThreadMxBean() method of Magagement Factory.

**ThreadMxBean bean= ManagementFactory.getThreadMxBean();**

2) After getting an instance of ThreadMxBean call **findMonitorDeadLocked** **thread()** method on it, IT returns on array of tryp long containing ids of all currently dead locked threads

Long ids[] = bean.findMonitorDeadLockThreads();

3) After getting ids of deadlocked threads, pass these ids to getThreadInfo()

Of ThreadMxBean .it will return an array of ThreadInfo objects, where one thread info object contains the details of one deadlocked thread.

ThreadInfo threadInfo[] =

bean.threadInfo(ids);

1. Iterate the threadInfo to get the details of individual deadlocked threads.

for(ThreadInfo threadInfo1 : threadInfo ){

s.o.p(threadInfo1.getThreadName());

}

Here are the some of Thread Info class which are used to retrieve the details of deadlock threads.

GetThread() -> returns ID of a dead locked thread.

GetThreadName()

-> Return the dead locked thread.

GetBlockedTime()

-> Returns the elapsed time in milli second that a thread is in dead lock.

GetLockName() -> Return string representing of an object for which thread has been waiting.

GetLockOwnerId()

-> Returns Id of a thread that currently owns the object.

getLockOwnerName()

-> Return the name of thread that currently owns the object lock.

**How to avoid the deadlock in java**

1) try to avoid nested synchronised blocks

synchronized (lock1) {

synchronized(lock2) {

//avoid this block.

}

}

1. Lock order

If we need nested synchronise blocks at any cost, then make sure that threads acquire the needed locks in some predefined order.

Example

If there are threads t1,t2,t3 running currently and these needed locks A,B and C in the following manner.

Thread1:

Lock A

Lock B

Thread 2:

Lock A

Lock C

THREAD 3

LOCK A

LOCK B

LOCK C

In the above senario t1 needs A, B Locks.

T2 need A and C locks

T3 need A B and C lock

If we define an order to acquire the locks like Lock A must be acquired before Lock B and Lock b must be acquired before Lock c, then deadlock never occurs.

By defining such lock ordering, then thread t2 never acquire Lock C and t3 never acquire Lock B and Lock C until they got Lock A. They will wait for Lock A until it is release by t1.

After Lock A is released by t1 Any one of these threads will acquire Lock A on the Priority basis and finished their tasks. Other thread which is waiting for Lock A will never try to acquire remaining Locks.

**Lock out time.**

Another dead lock preventive tip to specify time for a thread to acquire lock, it it fails to acquire the specified lock in the given time, then it should give up trying for a lock and retry after sometime. Such method of specifying time to acquire the lock is called lock timeout.

-> Lock the code when it is actually needed for example, if you want only some part of the method to be thread safe then lock that part not the whole method

Void method() {

//some statements

Synchronized(this) {

// locking only some part of the methods.

}

}

-----------------------------------------------------------------------------------------------

**Callable and Future**

**Callable :** Is simpler to Runnable interface

java.util.Consurrent.Callable

interface Callable<v> {

public abstract v call() throws Exception;

}

It will return any object and also throws Exception.

The Executor farmwork offers submit method to execute Callable implantations in threadPool.

The java Executor framwork follows **WorkerThreadPattern**.

As you may remember in java, a runaable acts as the target of a thread and in runnable interface a **public void run()** method has to implemented where you define the task, which will be executed by threads in thread pool. The Executor frame work assign work to threads only if there is an unavailable thread with pool. If all threads are in use, the workhas to wait once a task is completed by the thread that thread returns to the pool as an available thread.

Callable is same as Runnable, but it can return any type of Object if we want a result or state from work.

Java Future

Java Collable taks returns java.util.Connurrent.Future

Objects java future provides a cancel() method to cancel the associated Callback task.-

There is on over loaded version of the get method, where we can specify the time to wait for the result.

Get() is a synchronized, until the callable finishes its tak and return a value, it will wait for a callable.

IsDone()

IsCancelled()

To find out the current status of associate callable task.

--------------------------------------------------------------------------------

JVM Shutdown Hook

Shutdown Hook are a special construct that allow developers to plug in a piece of code to be executed when JVM shutting down

This comes in handy in cases where we need to to special clean up operations in case VM shutting down.

public class ShutDownHook{

Public static void main(String args[]) {

Runtime.getRuntime().addShutdownHook(

new thread() {

Public void run(){

s.o.p(“shutdown hook is running”);

}

}

);

Sop(“Application Termination.”);

}

}

Once shutdown sequence starts it can be stopped by Runtime halt() only

Inter thread communication using wait(), notify() notifyAll()

Threads can communicatee with each other sing wait(),notify() and notifyAll() methos.

These methods are final method of java.lang.Object class

Public final void wait() throws InterruptedException

Public final void notify()

Public final void notifyAll()

Wait() This metho tells the currently executing thread to release the lock of this object and wait until some other thread acquires the same lock and notify it using either notify() or notifyAll() methods.

This method will throw checked exception if waiting thread is interrupted.

Notify() : This method wakes up one thread randomly that called wait() method on this object

NotifyAll() This method wakes up all the threads that called wait() method on this object

But only one thread will acquire lock of this object depending upon the priorities.

***These 3 ( wait, notify, notifyAll ) method must be called within synchronized method or block. Any thread which calls these methods must have lock of that object.***

Example: ThreadWaitExample

Here Thread T1 and Thread T2 sharing Shared Object.

Thread T1 is calling methodOne() and thread t2 is calling methodTwo() of S object. Both the methods are synchronized. That means for only threads to enter these methods, they must acquire lock of S object.

First thread t1 acquired the object lock and enters methodOne().

Thread 2 waits for thread1 to release the object lock. Thread t1 call wait() method with in methodOne() as soon as it calls wait() method it releases the lock of s object and goes for wait. Thread t2 acquires this lock and enters methodTwo(). After entering methodTwo() thread t2 sleeps for 5 seconds and call nofity() methodTwo() after entering methodTwo() thread t2 sleeps for 5 seconds and call notify() method on this object it wokes up thread t1, which is waiting for this object lock. And execute remaining statement in methodOne in this manner t1 and t2 communicated with each other and shared locks.

Things to remember about wait(),notify() and notifyAll()

1) If a thread calls notify() method and more than one thread are waiting for the object lock, then only one thread will be notified randomly.

2) When a thread call notifyAll() method on an object it notifys all the threads which are waiting for the object lock, But only one thread will acquire this object lock depending upon priority.

3) When you call sleep() method on a thread, thread goes to sleep with holding the object lock with in it. But if we call wait() method , thread releases the object lock and goes for sleep(). This is the main difference between wait() and sleep() methods

4) wait() notify(0 notifyAll() are final methods of object

1. Wait(),notify() and notifyAll() will throw IlligalMonitorStatException. If calling thread does not owns objects lock.
2. Wait() is overloaded.

**Thread interruption in java**

Public void interrupt()

Public static boolean interrupted()

Public boolean isInterrupted()

Is a mechanisam in which a thread which is either sleeping or waiting can be made to stop sleep or waiting.

Thread interruption is like telling the thread that it should stop waiting or sleeping and return to running state.

Thread interruption is Programmatically implemented using interrupt() method of java.lang.Thread class.

The whole thread interruption mechanism depends on internal flag called interrupt states

The initial value of this flag for any thread is false

When you call interrupt() on a thread interrupt status of that thread will be set to true.

When a thread throws InterruptException, this status will be set to false again.

Wait(),sleep(), join() will throw InterruptedException

We can check whether a thread is interruped or not by using isInterrupted() method

There is a one more method to check interrupt status of a thread call interrupted() method. It is a static method of a thread class

It also returns the current states of a thread like isInterruped() method.

But it clears interrupted status of a thread I.e if interrupt status of a thread is true , their it will set the status to false.

Interrupt method will throws SecurityException, if current thread can’t Interrupted a calling thread.

**Thread life cycle or thread states in java**

There are SIX Thread states in java

NEW

RUNNABLE

BLOCKED

TIME-WAITING

TERMINATED

At any point of time threads will be in following states

Exmaple Thread states[] states = Thread.state.value();

For(Thread.state state : states) {

sysout(state);

}

Output will be above state

1 NEW

Thread t = new Thread();

2 Runnable

T.start

1. Blocked a thread will be in this state when a thread is waiting for object lock to enter into synchronized method/block
2. WAITING
   1. A thread will be in this state when wait() or join() method is called.
3. TIMED\_WAITING
   1. Sleep()

6) TERMINATED

When a thread get job done

**Thread Group in java**

Thread group in java is used to group similar threads into one unit.

Thread group are constructed using java.lang.ThreadGroup.

Main use of thread Group is we can handle multiple threads simultaneously.

How to add a thread group

ThreadGroup parentGroup = ThreadGroup(“parent thread group”);

Thread t1 = new Thread(parentGROUP, “THREAD-1”);

Thread t2= new Thread(parentGROUP, “THREAD-2”);

Some thread method

SetDaemon()

IsDaemon()

SetMaxPriorityes()

GetMaxPriority()

ActiveCount() : returns the number of active threads in specified group

Ex: ParentGroup.activeCount() o/p 3

ActiveGroupCount()

Returns the numbers of active thread group in specifed group

Interrupted() this method is used to interrupt all threads in group

Destroy()

This method is used to destroy whole group

Enmuerate() method

Public int enumerate(Thread list)

-It copies all active threads of a group into specifed array of threads

**How to stop a thread**

There are two ways through which we can stop a thread in java

Boolean variable

Interrupte()

Using boolean variable :

Define/declare a flag and assign a default value as true

Keep task run in while loop inside the run method by passing the flag, this flag make thread continue to run until flag becomes false.

Define stopRunning() this method will set the flag as false and stop the thread.

Declare flag as volatile ( this makes thread to read its value from the main memory, this making sure that thread always gets it’update value

Example : ThreadStopUsingBooleanFlag.java

Using interrupt() method

When ever we call interrupt() method on a thread, it sets interrupted of a thread, This status can be fetched by interrupted method

This status is used in a while loop to stop a thread

Block vs waiting states in java

|  |  |
| --- | --- |
| WAITING | BLOCKED |
| The thread will be in this state when it calls wait() or join() methods. The thread remains WAITING state until any other thread calls notify() or notifyAll()  The WAITING thread can be interrupted | The thread will be in this state when it is notified by other thread but has not got the object lock yet  The BLOCKE IS thread can’t be interrupted. |

-----------------------------------------------------------------------------------------------

Java.lang.Runtime

Java application has a single instance of class Runtime that allows the application to interface with the environments in which the application running.

The current runtime can be obtained from the getRunTime method.

Using Runtime class we can start chrome browse.

Runtime.getRuntime() .exec(“google-chrome”);

Public void exit(int status)

This method terminates the currently running java V.M

Runtime.getRuntime().exit(0);

Runtime.getRuntime().gc();

Runtime.getRuntime().halt();

Thread pool in java :

A Thead pool reuses previously created thread to execute current tasks and offers a solution to the problem of thread cycle overload and resource thrashing. Since thread is already existing when the request arrives, the delay introduced by thread creation is eliminated, making application more responsive.

Java provides Executor Frame work, which is centred around Executor interface, its sub interface Executor service and the class ThreadPoolExecutor which implements both of these interface. By using executor, one only has to implement to Runnable objects and send them to the executor to execute.

To use thread pool, we first create a object of ExecuteService and pass set of tasks to it. ThreadPoolExecutor class allows to set the core and maximum pool size, The runnable that are run by a particular thread are executed sequentially.

**Methods**:

newFixedThreadPool(int) - Create a fixed sized thread pool.

newCachedThreadPool() - Creates a thread pool that creates new threads as needed, but will resuse previously constructed threads when they are available.

newSingleThreadExecutor() - Creates a single thread.

Example :

1) Create a task (RunnableObject) to execute

2) Create Executor Pool using executor.

3) Pass the task to Executor Pool.

4) shutdown the executor Pool.

Example : in git

Risks in using Thread Pools :

1 **DeadLocks :** Thread Pools introduced another case of deadlock, one in which all the executing threads are waiting for the results from the blocked threads waiting in the queue due to the unavailability of threads for execution.

**2) Thread Leakage**  Thread leakage occurs if a thread is removed from the pool to execute a task but not returned to it when the task completed. As an example, if a thread throws an exception and pool class does not catch this exception then the thread will simply exit, reducing the size of the thread pool by one. If this repeats many times, then the pool would eventually became empty and no thread would be available to execute other request.

**3) Resource Thrashing :** if thread pool size is very large then, time is wasted in context switching between threads. Having more threads than the optimal number may cause starvation problem leading to resource thrashing as explained.

**Semaphore in java** :

A semaphore controls access to a shared resource through the use of counter. If counter is greater than zero then access is allowed, if it is zero then access is denied.

This to access the resource, a thread must be granted a permit from the semaphore.

**Working with semaphore :**

In general to use a Semaphore, the thread want access to the shared resource tries to acquire a permit.

If the semaphore count is greater than zero then the thread acquires a permit, which causes the semaphores count to be decremented.

Otherwise the thread will be blocked until a permit can be acquired.

When the thread no langer needs as access to the shared resource, it releases the permit which causes the semaphores count to be incremented.

If there is another thread waiting for a permit then the thread will acquire a permit at that time.

Java provides semaphore class in java.util.Concurrent page ( So we don’t need to implement our own semaphores)

Constructors in semaphore class.

There are two con0stants in semaphore class

Class

Semaphore(int num)

Semaphore (int num, boolean how)

Here num specifies the initial permit count

Num specifies the number of threads that can access a shared resource at any one time.

If it is one, then only one thread can access the resource at any time.

**Using semaphore as Locks (Preventing race condition)**

We can use a Semaphore to lock access to a resource, each thread wants to use that resource must first call **acquire()** before accessing the resource to acquire the lock, When thread done with the resource it must call release() lock.

Example 1

**Count Down Latch**

Count down latch is a synchronization id that allows one or more threads to wait until a set of operations being performed in other threads completes.

This class enables a java thread to wait until other set of threads completes their tasks.

Example : Application main thread want to wait till other service threads which are responsible for starting farmwork services have completed started all services.

CountDownLatch works by having counter initialized number of threads, which is decremented each time a thread complete it execution, When cout reaches to zero, it means all thread have completed their execution and thread waiting on latch resume the execution.

Example :

Main thread start

Create countDownLatch for new threads

Create and start new threads

Main thread wait on latch

‘N’ threads complete there task and retunrs

Main thread resumes execution.

Advantage

We can achieve Parallelism

**Cyclic Barrier**

Is another concurrent utility introduced in java 5

When number of threads want to wait for each other at a common point also

Know as the barrier before start processing again.

It is similar to CountDownLatch, but instead of calling countDown(), each thread calls a wait(). And last thread calls a wait() which signals that it has reached the barrier. All thread started processing again, also known as barrier is broken.

Usage : writing a unit test for concurrent programming, to simulate concurrency in a test class.

Example :

**Yeild() :**

Yeild means let go, to give up, to surrender.

A Yeild thread tells the virtual machine that it’s willing to let other threads to be scheduled in it place.

Yield is a static method and Native.

Yield tell the currently executing thread to give a chance to the thread that have equal priority in thread Pool.

It can only make a thread from running state to Runnable state.

Example :

Generics :

Generics are introduced in java 5 to provide the type checking at compile time.

If you are using Generics, you need not to perform the casting explicitly.

Java compiler applies strong type checking at compile time. If you use generics, you need not to perform the type casting explicitly.

Java compiler applies strong type checking if you use generics in your code and shows errors if the code violets the type safety. Thus, removing the rick of ClassCastException.

Therefore, to write type safety code and to remove the rick of ClassCastException at run time we need generics.

**How to define our won generics**

The syntax for defining generics class is as follows

Class class-name<T1,t2,….Tn> {

//Generic type or Parameterized type

}

Where T1,T2,T3 Stands for type

1. Enclosed in <> are called type parameters
2. And class ‘Class-Name’ is called generic type or parameterized type

Class GenericType<T>

{

T t;

Public GenericClass(T t) {

This.t = t;

}

Public void setT(T t){

This.t=t

}

Public T getT(){

Return t;

}

}

While creating as instance to the above generic class you can pass any class type as a type parameter and that class type replaces generic T for that object.

Public class GenericInJjava{

Public static void main(String args[]) {

GeneriClass<String> gen = new GenericClass<String>(“It must changed”);

String s = gen1.getT();

Gen1.setT(NEW iNTEGER(123)) //compile time error.

}

}

***Generics works only with Derived types***

While creating an instance of generic class you must pass only derived types. We can’t pass primitive types. If you pass primitive type, it gives compile time error. I.e generics works with derived types.

***Object of same Generics class Differ Based on their type Parameter***

Object of same generic class differ depending upon their type parameter.

For example, object of above generic class created using string type is not compatible with object of same class created using Integer type.

Generic<String> gen1 = new Gc<String>(“value of t”);

GC<Integer> gen2 = new GC<Integer>(new Integer(123));

Gen1=gen2 // Erorry type mismatch

Gen2 = gen1 // error

***Generic class with two type Parameters***

Class GenericClass<T1,T2> {

T1 t1;

T2 t2

}

We can create own type which creating generics

Class GenericClass<T> {

T t;

Pubic GenericClass(){

This.t = t;

}

Public void setT(T t) {

this.t = t;

}

Public T getT(){

Return t;

}

}

Class A{

Int I;

Public A (int i) {

This.i =I;

}

Public class GenericInJava {

Public static void main(String[] args) {

GenericClass<A> gen1 = new GenericClass<A>(new A(10));

GenericClass<A> gen2 = new GenericClass<A>(new A(10));

Sop(gen1.getT().i);

Sop(gen1.getT().j)

}

}

***Rules to follow while Implementing Generic Interface***

1. Only generic class can implement generic interfaces. Normal class can’t implement generic interface

Class GC<T> implements GI<T>{

---

}

Not like below

Class NormalClass implement GenericInterface<T> {…}

***A normal class can implement a generic interface, if type parameter of generic interface wrapper class.***

For example, below implementation of generics interface is legal

Interface GenericInterface<Integer> {

//Generic interface with integer as type parameter.

}

Class NameClass implements GenericInterface<Integer>

{

// Normal class implementng generic interface.

}

***Class implementing generic interface at least must have same number and same type of Parameters and at most can have any number and any type Parameters***

Interface GenericInterface<T>

{

//generic interface with one type parameter

}

Class GenerClass<T> implements GeneriInterfac<T> {

//class with same type parameter

}

Class GenericClass<T,V> implements Generic<T> {

// class with two type parameters

}

Class GenericClass<T1,T2> implementsGeniric<T> {

//compile time error, class having defferent type of parameter.

}

***You can change the type of parameters passed to generic interface while implementing it.***

When changed the class which is implementing should have new type as parameter and also you have to change old type with new type while implementing the method.

Interface GenericInterface<T> {

Void setT(T t) {

T getT();

}

}

Changing the type of parameter passed to generic interface while implementing

class GenericClass<V> implements <V> {

V t;

@override

Public void setT(V t) {

This.t= t;

}

@override public void getT(){

Return t;

}

}

***Generic interface can have any number of type parameters class implementing generic interface at least must have same type parameters and at most can have any number of parameters***

Interface GenericInterface<T1,T2,T3,T4,T5>

Class GenericClass <T1,T2,T3,T4,T5> implements GenericInterface <T1,T2,T3,T4,T5>

Class GenericClass <T1,T2,T3> implements GenericInterface <T1,T2,T3,T4,T5> {

//compile time error must have same number of type parameters.

}

Class GenericClass<t1,t5,t6> implement<t1,t2,t3,t4>{

///compile time error must have same type of parameters

}

***Class can implement more than one generic interface class should have type parameters of both the interface***

*Interface GenericInterface<T>*

***{***

*//Generic interface with one type parameter*

*}*

*Interface GenericInterface<T2,T3> {*

*//Generic interface with two type parameters*

*Class GenericClass <t1,t2,t3> implements Gi<t1>,gi<t2,t3>{*

*// class having parametr of both the interface*

***One more additional to generic interface, if you don’t want whole class or interface to be generic, you want only some part of class as generic, then generic method will be a solution for this.***

*Syntax*

*<Type-parameter> return-type method-name() {*

*….*

*}*

*<type-parameter> is defined just before return type*

***Generic methods can be static or non-static***

***Generic class as well as non-generic class can have generic methods.***

***Here is an example which contain static generic method defined inside a non generic class****.*

*Class NonGenericClass {*

*static <T> void genericMethod(T t1) {*

*T t2 = t1;*

*S.o.p(t2);*

*}*

*}*

*While calling generic method we can pass any type parameters*

*NongenericClass genericMethod(“String”);*

*NongenericClass getGenericMethod(new Double(11.11));*

***Constructors as Generics.***

*Like methods, constructors also can be generic*

*Even nongenetic class can have generic constructors*

*Here is an example of non generic class with generic constructor.*

*Public class GenericClass {*

*Public static void main(String args[]){*

*NonGenericClass nonGEn = new NonGenericClass();*

*NonGenericClass nonGen2 = new NonGenericClass();*

*}*

*}*

*Class NonGenericClass {*

*Public <T> NonGenericClass(T t) {*

*T t1 = t2;*

*S.OP(t1);*

*}*

*}*

***Bounded Types***

*Using bounded types, you can make the objects of Generic class to have data of specific derived types.*

*For example if we want generic class works only with numbers(int double,float....) then declare type parameter of that class as bounded type to java.lang.Numbers. Then while creating object to class you have to pass only Number types or its sub class type as type parameter.*

*Syntax for bounded types*

*<T extends SuperClass> I.e <Integer extend Number>*

*Here T can only replaced by superclass or its sub class*

*Class genericClass< T extends Number>{*

*T t;*

*Public GenericClasss(T t) {*

*This.t = t;*

*}*

*Public T getT() {*

*Return t;*

*}*

*In this example T is declared to bounded to number.*

*Voilation of this will throw compile time error.*

*Bounded type Parameters in Generic class*

*Public stati <T extends Number>*

*void PrintNumber<T T)*

***Using Interface as an upper bound***

*Class GenericClass<T extends anyClass & first interface & second interface>{*

*---*

*}*

***Wildcard Arguments***

*Mean unknow type arguments*

*They just act as place holder for real arguments to be passed while calling method.*

*They are denoted by question mark (?)*

*One important thing is that the types which are used to declare wildcard arguments must be generic types.*

*Wildcard arguments with unknow type*

*Wild card Arguments with an upper bound.*

*Wild card argument with Lower bound.*

*Static void processElements(ArrayList<?> a){ …}*

***Wild card Arguments with An Upper bound***

*GenericType<? Extends SuperClass>*

*This specifies that a wildcard argument can contain SuperClass type or it’s subclass. Remember that extends clause is an inclusive bound I.e superclass also lies in the bound.*

*Static void processElement(ArrayList< ? Extends Number> a)*

*Note : ‘Super’ clause is used to specify that lower bound for only wild card arguments it does not work with bound types.*

***Generic And their Interface***

*You have to follow some rules while making generic class as a super class or sub class.*

*A generic class can extend a non generic class.*

*Class NonGenericClass{*

*// NonGeneric*

*}*

*Class GenericClass<T> extends NonGenericClass {*

*//Generic class extending non generic class.*

*}*

***Generic class can also extends another generic class***

*When generic class extends another generic class, sub class should have at least same type and same number of type parameters and at most can have any number and type of parameters.*

*Class GenericSuperClass<T> {*

*//Generic class with one type parameter*

*}*

*Class Genericsubclass<T> extends GenericSuper {*

*// sub class with same type parameter.*

*}*

*Class GenericSubClass<T,V> extends GenericSuperClass{*

*// subclass with two type parameters*

*}*

*Class GenericsubClass<T1,T2> extends GenericSuperClass<T>{*

*// compile time error, sub class having different type parameter.*

*}*

*When generic class extends another generic class, the type arguments are passed from subclass same as in the case of constructor chaining, where super class constructor is called sub class constructor by passing required arguments. For example in below program T in Generic superclass will be replaced by String.*

*Class GenericSuperClass(T t) {*

*T t;*

*pubilc GenericSuperClass(T t) {*

*This.t =t*

*}*

*}*

*Class GenericSubClass<T> extends GenericSuperClass<T> {*

*Public GenericSubClass(T t) {*

*Super(t);*

*}*

*}*

*Public class GenericClassInJava{*

*Public static void main(String args[]) {*

*GenericSubClass<String> gen = new GenericSubClass<String>(“i am string”);*

*s.o.p(gen.t);*

*}*

*}*

*}*

***A generic class can extend only one Generic class and one or more generic interface. That it’s type parameter should be union of type parameter of generic class and generic interface****.*

*Class GenericSuperClass<T1>*

*{ ../}*

*Interface GenericInterface<T1,T2> {}*

*Interface GenericInterface<T3,T2>{}*

*Class GenericClass <T1,T2,T3> extend GC<T1> implements GenericInterface <T1,T2> , GenericInterface < T3, T2 >*

***Non-Generic class can’t extend generic class except of those generic class which already predefined types as their type parameters.***

*Class GenSupClass<T>{*

*}*

*Class NonGenericClass extends Generic SuperClass {*

*//compile time error, non generic class can’t extends generic class.*

*}*

*Class A { //predefind class }*

*Class GenSup class<A> {*

*// Generic class with predefind ‘A’ as typparamenter*

*}*

*Class nonGenClass extend GenSuperClass<A> {*

*// no compile time erroo // it is*

*}*

*Non-Generic class can extend generic class by removing the type parameter I.e as row type but it gives awaring.*

*Class GenericClass<T> {*

*T t;*

*Public GenericClass(T t) { this.t = t; }*

*Class NonGenericClass extends GenericClass {*

*Public NonGenericClass(String s) {*

*Super(s);*

*}*

*}*

*Public class GenericInJava{*

*Public static void main(sTring arg[]) {*

*NonGenericClass nonGen =*

*new NonGenricClass(“i am string”);*

*sop(nonGen);*

*}*

*}*

*While extending a generic class having bounded type parameter type parameter, type parameter must be replaced by either upper bound or its sub class.*

*Class GenericSuperClass<T extends Number> {*

*//generic super class with bound type parameter*

*}*

*Class GenericSubClass extends GenericSuperClass<Number>*

*{*

*// type parameter replaced by upper bound*

*}*

*Class Genericsubclass extends GenericSuperClass<T extends Number>{*

*//compile time error*

*}*

***Generic method of super class can be overridden in the sub class like normal methods.***

*Class genericClass {*

*<T> void genericMetod(T t) {*

*Sop(t);*

*}*

*}*

*Class NonGenericClass extends GenericClass {*

*<T> void genericMethod(T t) { sop(t)}*

*}*

*Public class GenericInJava {*

*Public static void main(string[] args) {*

*new GenericClass().genericMethod(“i am string”);*

*new NonGenericClass().genericMethod(“i am string”);*

*}*

*}*

***Type Erasure***

*One more instrusting thing about generics is type erasure. When you compile your java code compiler removes all generic information mentioned in code. Compiler replaces all type parameters with their bounded type. The Type parameters which don’t have bounds will be replaced with java.lang.Object class. That means all type parameters exist still compilation only. They are erased during compilation They don’t exist at run time.*

*Class Generic classOne<T> { T t;}*

*Class GenericClassTWO<T extend Number> { T t;}*

*After compilation code will look below*

*Class GenericClassOne extends java.lang.Object*

*{*

*Java.lang.Object t;*

*}*

*Class GenericClassTwo extends java.lang.extends*

*{*

*Java.lang.Number t;*

*}*

*You can notice that type parameters are erased after compilation. They don’t exist at run time. That is why yoy cannot instance a type parameter , it gives compilation error.*

*Class GenericClass<T>{*

*T t = new T();*

*<V> void generiMethod(){*

*V v = new V(); // Compile time error*

*}*

*}*

***Some intrusting observations about Generics in java***

Java allows generic class to use without type parameters i.e as a

“row type”. This is because to provide the compatability of generic code with non-generic code that means, non-generic code must be able to work with generic code and generic code must be able to work with non-generic code.

Class GenericClass<T>{--}

Public class GenericInJava{

Public static void main(String args[]) {

GenericClass rawtype = new GenericClass();

}

}

*You can’t create an instance to the type paramete. This is because, the type parameter does not exist at runtime. They are erased during compiletime.*

*Class GenericClass<T>*

*{*

*T t = new T(); // compile time error*

*<V> void genericMethod() {*

*V v = new V(); //compile time error*

*}*

*In general class with type parameter T, you can’t declare ‘static fields of type T’ and you can’t use T in a static method. However, you can declare static generic methods with their own type parameters.*

*Class GenericClass<T> {*

*Static T t; // Compile time error*

*Static void staticMethod(){*

*Sop(t); // compile time error*

*}*

*Static <V> void genericMethod(){*

*// static generic method.*

*}*

*}*

*You can’t instantiate an array whose type is a type parameter*

*Class GenericClass<T>{*

*T[] t;*

*Public GenericClass(T[] t){*

*T = new T[] // compile time error*

*This.t t;*

*}*

*}*

*You can’t create an array of Generic type containing specific type of data. But you can create an array of generic type containing unknow type of data.*

*Class GenericClass<T> {*

*//Generic type*

*}*

*Public class GenericsInJava{*

*Public static void main(){*

*GenericClass<Number> gen[]= new GC<Number>[10];*

*// compile time error.*

*GC<?> GEN[10] = new GC<?>[10] // This is fine*

*}*

*}*

*You can’t create generic exception I.e a generic class can’t extends throwable or any of its sub classes.*

*Class GenericClass<T> extends Throwable{*

*//compile time error*

*}*

*With unknow type we can’t create ( or ) instanciate an type parameter, but we can instanciate a know type.*

*T = new T[5]; //Compile time error*

*GenericClass<Number> gen[] = new GenericClass<Number>();*

*// Compile time error*

***Comparable and Comparators.***

*Java provides two interface to sort objects using data members of the class*

***Comparable***

***Comparator***

***Java.lang.Comparable.compareTo()***

***Java.util.Comparator.compare(Object1, Object2)***

*To summarize if sorting of objects needs to be based on natural order then use Comparable, where if you want sort need to be done on attributes of different objects, then use comparator in java.*

***Comparator:*** *is external to the element type we comparing, it’s a separator class, we create multiple separate classes (that implement comparator) to compare by different members.*

*Collections class has a second sort() method and it takes Comparator, The sort() method invokes the compare() to sort objects.*

*Create a class that implements Comparator.*

*Make an instance of the Comparator class*

*Call sort() method, give both the list and instance of class that implement Comparator.*

*Class Move implements Comparable<Move> {*

*Public int compareTo(Movie m) {*

*Return this.year-m.year;*

*}*

*}*

*Class Rating\_Comparator implements Comparator<Move>*

*{*

*Public int compare(Movie m1, Movie m2){*

*If( m1.getRating() < m2.getRating() ){*

*Return -1*

*}*

*If(m1.getRating > m2.getRating()) {*

*Return 1;*

*}*

*}*

*}*

*Class NameComparator implements Comparator<Movie>{*

*Public int compare(Movie m1, Movie 2){*

*Return m1.getName().compareTo(m2.getName());*

*}*

*}*

*Class Main {*

*Public static void main( String[] args) {*

*ArrayList<Movie> list = new ArrayList<Movie>();*

*List.add(new Movie(“future is here ”,3, 1995));*

*S.o.p(“Sorted by rating”);*

*RateCompare ratingCompare = new RateCompare();*

*Collection.sort(list, ratingComparators);*

*S.O.P(“sort by name”);*

*NameComparater nameComparator =*

*new NameComparater();*

*Collection.sort(list, nameComparator);*

*}*

***Serial soft V/S Parallel sort in java***

*Array element can be sorted by using sor() method.*

*Sort() method will use merge sort or TimSort.*

*Java 8 there is a new API introduced for Sorting which is paraller sorting.*

*Parallel Sort used Fork/join farmwork introduced in java 7 to assign the sorting task to multiple threads available in the Thread pool.*

*Arrays.sort() - is a sequential sorting*

*Arrays.ParallelSort() is a parallel soring.*

*Example : Need to work on this.*

***How to create Immutable class in java***

*Immutable class means once an object is create we can’t change its content*

*In java all Wrapper class and String are immutable.*

*We can create own immutable class as well*

*Class must be declared as final*

*DataMember is class must be final*

*A parametarized Constractor*

*Getter method for variable*

*No setter methods*

***InstancOf operator VS isInstance()***

*InstanceOf operator and isInstance() method both used for checking the class of object. But main difference comes when we want th check the class of object dynamically, in this case isInstance() will work.*

***Java virtual Machine***

*Jvm loads the class files and interprets the byte code to mechine code.*

***Searching Algorithm***

*Linear Serach O(n)*

*Binary Search O(logn)*

*Recursive Linear Serach O(n)*

*Recursive Binary search O(log n)*

***Sorting Algorithm***

*SelectionSort O(N2)*

*InsertionSort 0(N2)*

*MergeSort O(N log n)*

*BubbleSort O(n2)*

*TimSort*

*Http Methods*

*GET (READ)*

*POST(CREATE)*

*PUT(REPLACE/UPDATE)*

*DELETE(delete record.)*

***WeakHashMap***

*If any object is specified as key, dons not contain any reference, it is eligible for garbage collection, even though it is associate with key.*

*WeakHashMap*

*It won’t implement clone() method.*

***CopyOnWriteArrayList***

*A Thread-safe variant of Array in which all mutative operation (e.g add, set, remove) are implemented by creating copy of list which is a different way than vector or other collection use to proved thread safe.* *It is useful when you can’t or don’t want to synchronize the traversal, you need to prevent interference among current threads.*

*It is very efficient when you have list and need to traverse over its elements and don’t modify often it.*

*Iterator does not throw concurrentModificationException even if copyOnWriteArrayList is modified once iterator is created, because iterator is iterating over the separate copy of Arraylist while Write operation is happening on anther copyof ArrayList.*

*Ti throws UnSupportedOperationException if we try to modify copyOnWriteArrayList through iterator’s own method.*

***ArrayList and LinkedList remove() methods in java***

*Remove(Object obj);*

*Removes the first occurrence of the specified element from given list, if element is present.*

*After removing, it shifts subsequent elements. If any to left and decrease their indexes by 1*

*Throws ClassCastException if the type of the specified element is incompatable with the Collection*

*Throws NullPointerException – if the type of the specified element is null and this Collection does not premit null element.*

*Unsupported OperationException if remove operation is not supported by this collection*

*Remove(int index)*

*After remove it, it shift subsequent elements to left and decrease their index by 1*

*If the list contains int types, then this method is called when an int passed*

*Throws IndexOutOfBoundException.*

***Assertion in java***

*An assertion allows testing the correctness of any assumptions that have been made in the program*

*Assertion is achieve using the assert statement in java*

*While executing assertion, it is belived to be true.*

*If it fails JVM throws an error name AssertionError*

*The Assert statement is used with boolean expression and can be written in two different ways.*

*Assert expression*

*Assert expression1 : expression2*

*Example*

*Int value = 15;*

*Assert value >= 20 : “underweight”*

*Sop(“ value is : ” + value);*

***Enabling assertion :***

*By default assertion are disabled we need to run the code as given*

*Java – ea Test*

*Java –enable assertion Test.*

***Disabling assertions***

*Java –da Test1 or*

*Java –disable assertion*

Example : AssertionErrorExample1

*Arguments to private methods Private arguments are provided by developer's code only.*

***Volatile:*** *Using volatile is yet another way of making variables thread safe.*

*Thread safe means that a method or class instance can be used by multiple thread at the same time without any problem.*

*Class SharedObj {*

*//change made to shared var in one thread my not immediately reflect in other thread.*

*Static int shared Var;*

*}*

*Suppose that two objects are working on sharedObj, it two threads run of different processors each thread may have its own local copy of shared variable. If one thread modifies it’s value, the change might not reflect in the original one in the main memory instantly. This depends on the write policy of cache. Now other thread is not aware of this modified value which leads to data inconsistency.*

*Below diagram shows that if two threads are run on different processors, then value of the shared variable may be differed.*

*Class SharedObject {*

*//volatile key word here makes sure that changes make in one thread are immediately reflect in other thread.*

*Static volatile int sharedVar = 6;*

*}*

*The values of volatile variable will never be cached and all writes and reads will be done to and from the main memory.*

*With synchronized we can achieve mutual exclusion and visibility of a method or block*

*Volatile is something like achieving synchronization on variable.*

***Volatile tells the compiler that the value of a variable must never be cached as its value my change outside of the scope of the program itself.***

*Exmpale : VolatileExample.java*

*Java 8 features*

*Stream in java*

*Introduced in java 8*

*Used to process Collection of objects*

***Different operations on Streams***

*Immediate Operations*

*is a lazily execute and return a stream as a result*

*Hence various intermediate operations can be pipe lined.*

*Map() -> is used to map the items in the Collection to other objects according to the Predicate passed arguments*

*List number = Arrays.asList(2,3,4);*

*List squre = Number.Stream().map(x->x\*x).collect(collectors.toList())*

*Filter() -> The filter method is used to select elements as per the Predicate passed.*

*List name = Arrays.asList(“reflection”, “collection”,”stream”);*

*List result = names.strea().filter(s->s.startWith(“s”)).collect(Collectors.toList());*

*Sort()*

*List name = ArrayList.asList(“reflection”, “Collection”, “stream”);*

*Name.stream().sort().collect(Collectors.toList());*

*Terminal Operations*

*List arrayList = Arrays.asList(2,3,5,4);*

*Int even = number.stream().filter(x->x%2=0).reduce(0, (a,b)->a+b);*

*Reduce method is used to reduce the elements of stream to a single value.*

*Collect The collect method is used to return the result of the intermidiate operation performed on the Stream.*

*ForEach() The forEach method is used to iterate through every element of the steam.*

*List number = Array.asList(2,3,4,5,8);*

*ForEach(y->sop(y));*

*Lambda Expression in java 8*

*Lambda expression basically express instances of functional interce( an interface with single abstract method is called functional interface). An example java.long.Runnable*

*Example*

*Interface function {*

*Void abstract fun(int x);*

*}*

*Class Test {*

*Public static void man() {*

*FunctionaInterface f= (int x) -> sop(2\*x);*

*f.abstrac(5);*

*}*

*}*

*Runnble task = () -> {*

*SOP(Thread.currentThread.getName()+”is running”);*

*}*

*Thread thread1 = new Thread(task);*

*Thread1.start();*

*Public static void main(){*

*New Thread().start();*

*}*

***Big Integer classes***

*Big Integer is used for mathematical operation which involves very big integer calculations, that are outside the limit of all available primitive types*

*Java.math.BigInteger.*

*HTTP runs on 80*

*HTTP runs on 443*

*FTP runs on 21*

*0 to 1023 are predefined ports.*

***Lightweight Directory Access Protocol. (LDAP)***

*Is a standard application protocol for accessing and maintaining distributed directory information service over an internet protocol.*

***Queue***

|  |  |
| --- | --- |
| *Pool()* | *To remove and return head element of the queue, if queue is empty then the method returns null* |
| *Remove ()* | *To remove and return head element of the queue, if queue is empty then runtime exception NoSuchElementException* |
| *Peek()* | *To return head element of queue. If queue empty, then their method returns null* |
| *Element()* | *To return head element of queue, if queue is empty then runtime exception NoSuchElementException* |
| *Offer()* | *To add an object into the queue.* |

***Priority Queue****.*

*Elements are inserted based on priority.*

*Priority order either default sorting order or natural sorting order.*

*Insertion order not preserved.*

*Duplicates are not allowed.*

*Default sorting order. (element are Homogenesis and comparable)*

*Null is not allowed, even as first element also.*

*Constructors :*

*PrioritesQueue q = PrioritesQueue(); //default 11 is the size*

*PrioritesQueue q = PrioritesQueue(size);*

*PrioritesQueue q = PrioritesQueue(size, Comparator);*

*PrioritesQueue q = PrioritesQueue(Sortedsets);*

*PrioritesQueue q = PrioritesQueue(Collection c);*

*Sorted Map*

*Map (I)*

*SorteMap(I)*

*FirstKey()*

*LastKey()*

*HeadMap()*

*FailMap()*

*SubMap(k10, k20)*

*Compatrator()*

*TreeMap (class)*

*datastructure for tree map*

*RED-BLOCK tree [Hybrid tree]*

*Insertion ar not preserved*

*null value accepted*

*Hetroges(defined comparator)*

*If we follow default sorting order Key must be Homogenies*

*and comparable (other wise we will get ClassCastException)*

*if we use Comparator, key not be homogeneous nor comparable.*

*Null acceptance*

*Put(null, ‘’x);*

*For non empty tree, if we enter null key, then we get NullPointerExcepiton*

*Empty tree Null key allowed*

*If we add second element, the we get NullPointerException*

*Jdk 1.6 null allowed*

*Jdk 1.7 null values not allowed*

***TreeMap***

*TreeMap t = new TreeMap()*

*elements are added default softed order*

*TreeMap t = new TreeMap(Comarator c);*

*TreeMap t = new TreeMap(Map m);*

*TreeMap t = new TreeMap(SortedMap m);*

*Linked HashMap*

*Used : Cached based application*

|  |  |  |
| --- | --- | --- |
| *Hashmap* | *LinkedHashMap* |  |
| *Data structure HashMap* | *HashTable + Linked List* |  |
| *Inspersion order not preserved* | *Insertion order preserved* |  |
| *Intraduced in 1.2* | *Intraduced in 1.4* |  |

*Identity HashMap*

*== reference or address comparison*

*Equal() Content comparison*

*Integer I = new Integer(10);*

*Integer i2 = new Intgeger(10);*

*Sop(I==i2) // false*

*Sop(I.equa(i2)); //true*

*IdentityHash map exactly same as HashMap*

*HashMap will use equal method to checkkey*

*IdentityHashMap uses == to compare keys*

*JVM uses equal() mehod to identify duplicat keys*

*WeakHashMap*

*HashMap doninate G.C*

*G.C dominates weak hashmap*

*NavigableSet and Navigable Map*

*1.6 enhancement in collection framework.*

*Navigable set*

*Collection(1.2)*

*Set (I)*

*SortedSet(1.2)*

*NavigableSet(1.6)*

*TreeSet (1.2)*

*If defines several method for Navigation purpose*

*Floor(e) [D,e] = e*

*Return hight element, which is lessthan an or equal to e*

*Lower(e) < e retrn hight element which is lesstha e [d]*

*Ceiling(e) >= e*

*Return lowest element which is greater than e [F]*

*Higher(e) >e*

*return lowest element which is great than e.*

*pollFirst() - remove and return first element.*

*poolLast() - remove and return last elment*

*desendingset()*

*it returns NavigableSet in revers.*

*NavigableMap*

*map(I) 1.2*

*SortedMap(I) 1.2*

*NavigableMap 1.6*

*TreeMap 1.2*

*it define several method for Navigation purpose*

*Defines the following method*

*FloorKey(e)*

*LowerKey(e)*

*CeilingKey(e)*

*HigherKey(e)*

*PollFirstEntry(e)*

*PollLastEntry(e)*

DescendingMap()*;*

*Example* TreeMapExample1.java

***Collection Utilities***

*Defines several utilities method for collection objects.*

*Collections.sort(list);*

*Sort*

*Seraching*

*Sorting*

*Lists elements of list*

*Public static List sort(List l)*

*Default sorting order ( natural )*

*Elements shoud be Homogenious and Comparable*

*List should not contain Null value*

*Public static void(List sort, comparator c)*

*Searching elements of list*

*Binary search algorithm(standard searching algoritm)*

*BinarySearch(List l, Object obj);*

*BinarySearch algorithem*

*List shoud be sorted*

*Int binarySearch(List l, Object obj); // here return type int is index*

*Unsuccessful search return insertions point.*

|  |  |  |  |
| --- | --- | --- | --- |
| *0* | *1* | *2* | *3 <-- index* |
| *a* | *c* | *d* | *f* |
| *-1* | *-2* | *-3* | *-4 <-- insertion Point* |

*Int binarySearch(list l, b) return -> -2*

*If list not sorted then we will get UnExpected results*

*Public static int binarySearch(List l, Object obj, Comparator C);*

*Note :*

*For list of ‘n’ elements, in case of binary search method.*

*Successful search result range 0 – n – 1*

*Unsuccesful search result rang -(n+1) to –1*

*Total result rang -(n+1) to n-1*

*Example*

|  |  |  |  |
| --- | --- | --- | --- |
| *-1* | *-2* | *-3* | *-4* |
| *A* | *K* | *Z* |  |
| *0* | *1* | *2* |  |

*Successful search Range 0 to 2*

*Unsuccessful search Range –4 to –1*

*Total Result Range –4 to 2*

*Reversing elements of list*

*Public static void reverse(List l)*

*Public static void reverseOrder(Comparator c)*

*Comparator c1 = Collections.reverseOrder(Comparator c)*

*Arrays*

*Array Class is an utility class, to define several utility methos for Arrays Object.*

*Sorting elements of Array*

*Public static void sort(PrimitiveArray[]) //default sorting order*

*Public static void sort(Object[] o)*

*-> Default sorting order.*

*Public static void sort(Object[] o, Comparator c);*

*Searching elements of array :*

*Array class defines the follwing binary search methods.*

*Public static int binarySearch(Primitive[] p, Primitivetarget)*

*Public static int binarySearch(Object[] a, Object target);*

*Public static int binarySearch(Object[] a, Object target, comparator c)*

*Note : All rules of Array class binarySearch method are exactly same collection class binerSearch methos.*

***Converting Array to List***

*Object[] a= Collection.toArry();*

*Public static List asList(Object[] o);*

*String[] = {“a”, “z”, “b”};*

*List l = Array.asList(s);*

*Strictly speaking this method won’t create an independent list object, for the existing array we are getting list view.*

*We are not getting list object, list view of array.*

*By using array reference, If we perform any change, automatically that change will be reflected to list.*

*Simillarly by using list reference if we perform any change that change will be reflected to the array.*

*By using list reference we can’t perform any operation, which varies the size. Otherwise we will get runtime exception saying UnsupportedException*

*Example :*

*L.add(m); -> RuntimeException -- UnsupportedOperationException*

*L.remove(l);*

*L.set(1,”N”); --> Update indexed element*

L.set(1, new Integer(10)); RuntimeExcepton -- Array StoreException

*By using List reference, we are not allow to replace with Heterogeneous object otherwise we will get RuntimeException saying ArrayStoreException*

*L.set(1, new Integer(10)); // ArrayStoreException;*

***Need of Concurrent Collections***

*Most of the traditional collections are not thread safe.*

|  |
| --- |
| *Vector*  *HashTable*  Collections.*SyncronizedList() --> Thread safe,Only one thread allowed to Collections.syncrozixedSet(); --> work*  *Collections.syncronizedMap() -->* |

*ConcurrentModificationException*

Example: *ConcurrentModificationExceptionExample*

***Properties***

*We can use java properties object to hold properties which are coming from Properties file.*

*Properties P = new Properties();*

*Both Key and value should be string.*

*Methods*

*String getProperty(String propertyName);*

*String setProperty(String PropertyName, String PropertyValue);*

*// if Property Name already present in property, the new Property Value will be replaced with old Property Value.*

*Enumation propertyNames();*

*Load(InputStream is) // to load properties from properties file into java Properties object.*

*Store(OutputStream as, String comment) // to store properties from java properties obj into properies file.*

***HashTable :***

*The underlaying data structure is* ***Hashtable***

*Insertion order is not preserved, and it is based on hashcode of*  *keys*

*Duplicate keys are not allowed and duplicate values are allowed.*

*Hetrogenous objects are allowed for key and vlues*

*Null keys are not allowed, for Null values are not allowed,*   *otherwise we will get RuntimeWxception, NullPointerException.*

*It implements Serializable and cloneable interface, but not RandomAccess*

*Every method Present in HashTable is Syncronized and hence HashTable objects are Threadsafe.*

*HashTable is bast choice if our frequent operation is retrival/search operation.*

*HashTable ht = new HashTable();*

*Default capacity 11*

*Default loadfactor 0.75*

*Hashset/HashMap -> 16*

*HashTable h = new HashTable( int initialCapacity);*

*HashTable h = new HashTable( int initialCapacity, float fillRation);*

*HashTable h = new HashTable( Map m);*

***ArrayList and Set Difference***

|  |  |
| --- | --- |
| *List* | *set* |
| *Duplicates are allowed* | *Duplicated are not allowed.* |
| *Inserstion order Preserved* | *Insertion Order not preserved.* |
|  |  |

***Queue***

*If we want represent a group of individual objects prior to prcessing then we should go for queue.*

*Usually Queue fallows first in first out order, but based on our requirement we can implement our own priority order.*

*Example :*

*Before sending a mail all mailid’s we have to store in some datastructure in which order we add mail ids., in same order only mail should be delivered for this requirement Queue is best choice.*

***Collctions Object one by one Iteration.***

*1 Enumatation (I) Legacy class*

*2 Iterator (I)*

*3) ListIterator (I)*

***Utility class***

*Collections (C)*

*Arrays (C)*

*HashMap uses an array in the backgound, each element in the array ia another data structure usuall a linked list or binary search , tree.*

*Doubled LinkedList*

*Red-Block tree*

*HashTable*

*IdentityHashMap uses ‘==’ operater than equals() method.*

***BlockingQueue***

*Public interfac BlockingQueue<E> extends Queue<E>*

***BlockingQueue*** *that additionally supports operations that wait for the queue to become non-empty when retrieving an element and wait for space to become available in the queue when storing an element.*

***Serilization***

*The process of serving object state to a file system.*

*The Process of converting an object from java supported to file supported form or network supported form.*

***Deserilization***

*The Process of reading sate of an object from file.*

*The process of converting file/newtwork supporting form to java supporting form.*

***Implementation***

*Using two stream, we can implement Serilization* ***FileOutPutStrea, ObjectOutPutStream***

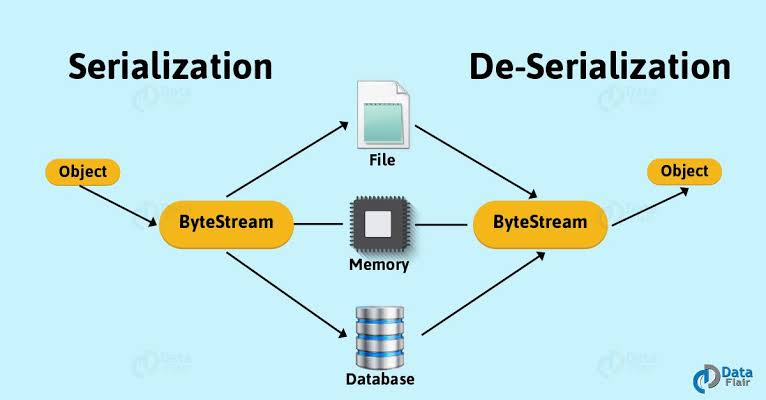
***ObjectOutputStream*** *: Take object and convert into binary data.*

***FileOutputStream*** *: takes binarydata and write into Filesystem.*

*FileOutputStream fos = new FileOutputStream(“abc.ser”);*

*ObjectOutputStream oos = new ObjectOutPutStream(fos);*

***Oos.writeObject(d1);***



***DeSerilization***

*Reading Object state from file system.*

***Implementation***

*Using two streams, we can implement de-Serilization* ***FileInPutStream*** *(Takes filepath, and converts it into binary data),* ***ObjectInPutStream****(takes binary data and converts into object state)*

*FileInputStream fis = new FileOutPutStream(“abc.ser”);*

*ObjectInputStream*

*Ois = new ObjectOutputStrem(fis);*

*Dog d = ois.readObject();*

***Object should implement Serizable interface, the only object can be de/seriable.***

***If not we will get*** *NotSerializableException*

***Customised Serialization***

*To recover loss of information because of transient keyword.*

*To do customisation we need to add some extra work at sender and reviver side*

*We need to add extra data while serilization and we need to extract data*

***Private void WriteObject(ObjectOutputStream oos) throws Exception***

***Private void ReadObject(ObjectInputStream fs) throws Exception.***

*These two methods are called as* ***callable*** *statement.*

*These two methods should be defined in Serializable object.*

***Example : CustSerializeDemo***

***Serilization with respect to inheritance***

*If parent is serializable then by default every child is serializable. That is Serializable nature is inheriting from parent to child. Hence even though child doesn't implement Sterilizable if parent class implements Serializable then we can serializable child class object.*

***Externalization***

*Serilization everything take care by JVM and programmer doen’t have any control.*

*It is not possible to save part of the object*

*The main advantage of Externalization over Sterilization is based on our requirement.*

*We can save either total object or part of the object, so relatively performance will be improved,*

*To provide externalizable, we need to implement externalizable interface*

*Java.io.Externalizable*

*Public void writeExternal(ObjectOutPutStream out) throws IOException*

*Public void readExternal(ObjectInputStream in) throw IOException*

*Every externalizable interace implemented class should compulsorily contain public no argument constructor.*

*If no public no arg constructor, JVM will through InvalidClassException*

*Example:* ***ExternalizableDemoMainClass.java***

|  |  |
| --- | --- |
| *Serializable* | *Externalizable* |
| *Serilizable is a maker interface* | *Externalization is have two methods*  *WriteExternal(ObjectOutputStream)*  *ReadExternal(ObjectInputStream)* |
| *No need of Public constructor* | *Must have public constuctor, otherwise, RuntimeException* |
|  | *InvalidClassException (R.E)* |
| *Transient play role* | *No role for transient.* |

***servialVersionUID***

*At the time of Serialization JVM save a unique id with every object. This unique id will be generated by JVM based on .class file. At the time of deserialization receiver side JVM will compare object unique id with local .class unique id.*

*If both are matching then only deserialization will be performed otherwise receiver unable to deserialize and will get RuntimeException saying InvalidClassException.*

*This unique identifier nothing but SerialVersionUID.*

***Problems***

*Both sender and receiver should use same JVM with respect to vendor and version.*

*To generate SerialVersionUID internally JVM will use complex algorithm which may create Performance Problems.*

*Should both sender/reciver class version should be same.*

***Solution to above problem***

*We can configure our own SerialVersionUID*

*Private static final long SerialVersionUID = 1l;*

***Transient (modifier)***

*Only for variable*

*Public class {*

*int i = 10;*

*transient j = 20;*

*}*

*Trasient variables don’t paticipate in Serilization process.*

*JVM simple ignore transient variables and replaces with default values*

***Static Vs Transient Vs final***

*static -> class level data*

*Serilization -> applicable to object*

*Static variables not part of Object state, if it is not part of object state, it won’t participate in Sterilization.*

*Declaring static variable as transient, there is not use.*

*Final variables are replaced by value at compile time only*

*final int a = 10;*

*int b = 20;*

*Transient is no use on final variable.*

|  |  |
| --- | --- |
| *declaration* | *output* |
| *Int I = 10*  *Int j = 20;* | *10 20* |
| *Transient int I = 10;*  *J = 20;* | *O,20* |
| *Transient static int I= 10;*  *Transient int j = 20;* | *10 0* |
| *Transient int I=10;*  *Transient final int j=20;* | *0, 20* |
| *Transient static int = 10;*  *Transient static int j =20;* | *10 20* |

***Annotations:***

*Annotation is a meta data, metadata is a data about data, So annotations are metadata for code.*

*@Override*

*@Target(ElementType.METHOD)*

*@Runtime(RetentionPolicy.SOURCE)*

*Public @interface Override {*

*}*

*@Documented : whether to put the annotation in java doc’s*

*@Retention When the annotation is needed*

*1) RetentionPolicy.SOURCE*

*// Discard during compile time*

*// These annotations don’t make any sense after the compile time*

*@Override,@supressWarrings*

***2)*** *RetentionPolicy.CLASS*

*Discard during class load*

*Useful when doing bytecode level post procession*

*3) RetentionPolicy.RUNTIME*

*Do not discard the annotation should be abailable for reflection at runtime.*

*@Target*

*Where the annotation can be place*

*If we don’t specify this, annotation can be placed any where.*

*ElementType.METHOD*

*ElementType.PARAMETER*

*ElementType.CONSTRUCTOR*

*ElementType.LOVALVARIABLE.*

*@Inheritad :*

*Controls whether the annotation should effect the subclass.*

***Ex :*** *AnnotationExample.java*

*If we have only one attribute inside an annotation it should be named “value” and can be used without attribute name while using it*

*@interface Author {*

*Strings.value();*

*}*

*@Author(“yash”)*

*Public void someMethod(){*

*}*

***Try with Resource***

*When the try block ends the resources are automatically released.*

*We no need to create a separate finally block*

*Try( BufferReader br = new BufferReader(new FileReader(“Filepath”)) ){*

*---*

*} catch(IOException e) {*

*e.printStackTrace();*

*}*

*How it works*

*Try with resource is available to any class that implements AutoCloseable interface.*

*In above example*

*BufferReader implements AutoCloseable interface*

*Public interface AutoCloseable{*

*Void close() throw Exception;*

*}*

***Static Initialization***

*We can use only static members of a class inside the static Initialization Block*

*Static initialization blocks are mainly used to initialize static fields of a class*

*Static initialization block is the first block to be execute after class is loaded in the memory*

*Static blocks are something like functions, they are stored in stack, they won’t store in memory*

*Static blocks cann’t be nested*

*Following Program gives Compilation error when execution*

*Class A {*

*Int I;*

*Static {*

*Sop(i);*

*}*

*}*

*Static initialization block is the first block to be executed after class is loaded in the memory.*

*Static blocks is mainly used to initialize static members of a class.*

*We can call static method inside static initialization block*

*We can’t call a non static method from static method it will be a compile time error.*

*Static blocks are executed before main method call.*

***Instance initialization block***

*While creating an object to a class, statements in instance initialization block will be executed first.*

*In any constructor, first statement will be either super() or this(), Second set of statements will be calling statements to all IIB’s of the class and*

*Third set of statements will be actual constructor .*

*Here is the template of a constructor*

*Constructor(){*

*super() or this()*

*[calling statems in IIB’s\]*

*[Statement of Constructors]*

*}*

***Constructurs***

*Constructors can’t be called recursively, it will be compilation error.*

*Local variable in java must get a value before they are used, otherwise compilation error*

*We can’t call super() and this() from a method*

*Constructor can’t have both super() and this()*

*Interface can’t be local member of a method.*

*InnerClasses*

*Local innerclass are not visible outside the method or block in which they are defined.*

*Anonymous inner class will not have name and you can instantiate anonymous inner class only once.*

*Static nested class can have static members and non static members*

*Member inner class conn’t have static member in them*

*Non static inner class must not contain static initializers block.*

*Local inner class can’t be declared with access modifiers (Private or Protected or public)*

***Difference between Iterator And List Iterator in java.***

*Iterator and List Iterator are two interface in java collection frame work.*

*Which are used to traverse the collection although List Iterator , there are some differences in the way they traverse the collections.*

*Using Iterator we can traverse list, Queue and set.*

*But using list Iterator we can only traverse List object.*

*Iterator Methods.*

*Boolean hasNext() checkes whether collection has more elements*

*E next() returns next element in the collection*

*void remove()*

*Remove the current element in the collection*

*Boolean hasNext()*

*E hasPrevious()*

*E next()*

*E previous()*

*Int nextIndex()*

*Int previousIndex()*

*Void remove()*

*Void set(E e)*

*Void add(E, e)*

*Using ListIterator you can obtain index of next/previous elements (this is not possible with Iterator)*

*Using ListIterator we can perform modification on the list (insert/update/remove). But using iterator you can only remove the elements from the collection.*

*From specified index we can traverse elements using ListIterator. Same is not possible with iterator*

***ArrayList VS LinkedList***

*Insert/update operationa are fast in LinkedList*

*Retrivel operation fast in ArrayList*

*Memory usage is more in LinkedList*

*Both maintain insertion order*

***Difference between sleep() and wait()***

*wait() and sleep(0 methods in java are used to pause the execution of a paticular thread in a multi thread environments.*

*When ever a thread call wai() method, it releases the lock or monitor it holds*

*When ever a thread calls sleep() method, it donn’t release the lock or monitor it holds.(it goes Timed waiting state)*

*Wait() method from object class , sleep() method from java.lang.Thread class*

*A Thread which is in WAITING state ( state after calling wait() method) can be wokenup by other threads by calling notify() and notifyAll() method.*

*But a thread which is in TIMED-WAITING state can not be wokenup ( if any thread interrup sleeping thread, interrupted exception will be raised)*

*Wait() along with notify and notifyAll are used for interthread communication, where as sleep() is used to pause the execution of current thread for a specified period of time.*

*Wait() is called on object, when ever it is called by a thread on a paticular object, thread releases the lock of that object and waits until other method threads call either nofify() or notifyALL() method on the same object, where sleep method is called on threads.*

*Whenever sleep() is called only current thread is going for sleep, it means thread calls sleep() on a thread t i.e t.sleep(), main thread itself is going to sleep not thread t.*

***Throw VS Throws VS Throwable***

***Throw*** *is a keyword in java , is used to throw an exception nanually.*

***Throws***

*Throws is a key word is used to specify in method definition saying that a method can capabile of trhowing an exception, it is the handler responsiblility to handle the exception*

***Throwable***

*Is a super class for any types of errors and exception in java*

*This class amember of java.lang package*

*Only instance of this class or it’s sub class are thrown by the java virtual mechine or by the throw statement*

*If you want to create your won customized exception, then your class must textedn this class*

*Class Myexception extends throwable { }*

*Class ThrowableExample {*

*Void method() throws MyException*

*{*

*MyException exc = new MyException();*

*throw e;*

*}*

*}*

***Difference between shallow copy VS Deep copy***

*Cloning is a process of creating an exact copy of an existing object in momory*

*Java.lang.Object clone()*

*Shallow Copy*

*Deep Copy*

*By default behaviou of clone method will be shallow copy*

*If Any object exits in cloning object, only reference will be copied in shallow copy, Where as in deep copy, value will be copied into new Object and reference will be pointing to new object.*

|  |  |
| --- | --- |
| *Shallow Copy* | *Deep copy* |
| *Clone object and original objects are not 100% disjoint* | *Cloned and original objeccts are 100% disjoint* |
| *Any changes made to Clone Object will be reflected in original as well as cloned Object* | *Any changes made to clone object will not reflect in original object vice versa.* |
| *Shallow copy is preferred in an object has only primitive fields* | *Deep copy is preferred if any object has reference to other object as fields* |
| *Shallow copy is fast and also less expensive* | *Deep copy is slow and very expensive.* |

|  |
| --- |
| *Throwable – class* |

|  |  |
| --- | --- |
| *Error* | *Exception* |

*Error : class represents the errors which are mainly caused by the environment in which application is running.*

*Recovering from Error is not possbile*

*OutOfMemoryError*

*StackOverFlowError*

*Exception class represent the exception which are mainly caused by the application itself*

*NullPointerException*

*ClassCastException*

*Exception are releate to application and error are related to Environment*

*Java.lang.Error*

*Java.lang.Exception*

***Difference between == and equals method***

*We can check the equality of an object in two ways*

*one is based on their location in the memory and second one is base on their content (i.e state of the object)*

*Both == and equal() method are used to check the equality of two objects*

*Equals() : method is ment to be used for comparison of two objects based on their contends*

*“==” Operator compares the objects based on their location in memory.*

***Difference between ClassNotFoundException VS NoClassDefFoundError***

*Both will be happened at runtime*

*ClassNotFoundException is an exception which occurs when you try to load a class at runtime using Class.forName() or loadClass() methods.*

*Class.forName(“oracle.jdbc.driver.OracleDriver”);*

***NoClassDefinitionFoundError***

*Is an error which occurs when a paticular class is present at compiletime but miss at run time.*

*Is any error which is thrown when java runtime system tries to load the definition of a class and class definition is no longer available.*

*The required class definition was present at compile time but is/was missing at runtime.*

*Class A{ }*

*Public Class B {*

*P s v m(){*

*A a = new A();*

*}*

*}*

*A.class*

*B.class are the two class files will be generated, delete A.class file and check for and runtime B.class file.*

***Program VS Process Vs Threads***

*Program Program is an executable file containing the se of instructions Witten to perform a specific job*

*Process Process is a small executable unit of program*

*Thread Thread is a smallest executable unit of process*

***Difference between user Threads and DaemonThreads***

***There are two type of threads***

***User Defined :*** *User defined threads are high priority threads which always run in foreground.*

*User threads are creaed by a application to perform some specific task, Where as daemon threads are mostly created by the JVM to perform some back ground tasks like garbage Collection*

*JVM will wait for user threads to finish their task, JVM will not exit until user threads finish their tasks. On the othe side JVM will not wait for daemon threads to finish their tasks.*

*User threads high prioriy threads, daemon threads are designed to elp user threads.*

*User threads are foreground threads and daemon threads are back ground thread.*

***Daemon threads*** *low priorities threads which always run in back ground.*

***Difference between statement VS Prepared statement VS Callable Statement.***

***Statement (I) :*** *used to execute normal SQL Query*

*Statement interface is used to execute normal SQL Queries.*

*Most of the time statement interface is used for DDL (Create,Alter and Drop) statement.*

*Statemetn stmt = Con.createStatements();*

*Stmt.execute(“create table student id number notnull”);*

***Prepared statement :*** *Used to execute dynamic or Parmetrized SQL Quries.*

*We can pass the parameter to SQL Queries at run time using this interface*

*It is useful to use Prepared statement if we are using executing a paticular SQL Query multiple times.*

*Prepared statement are Precompiled and the query plan is cread only once irrespective of how many times you are executing that query.*

*PreparedStatment pstmt = con.preparedStatment(“update STUDENT set NAME = ? where id = ?”);*

*Pstmt.setString(1,”NAME”);*

*Pstmt.executeUpdate();*

***Callable Statement :*** *Used to execute Stored Procedures*

*Is used to execute the stored Procedures*

*Callable statement extends Prepared statement*

*Using callable statement we can pass 3 types of Parameter to stored procedures.*

*IN,OUT,INPUT*

*CallableStatement cstmt = con.prepareCall(“{call Procedure\_name(?,?,?)}”)*

*Cstmt.execte();*

*It can be used to execute functions.*

***Extends Thread VS Implement Runnable in java***

*There are two way we can create/implement thread in java*

*One by extending java.lang.Thread - class*

*java.lang.Runnable -- interface*

|  |  |
| --- | --- |
| *Runnable* | *Thread* |
| *You can extend any other class* | *You can't extend any other class* |
| *No overhead of additional methods* | *Overhead of additional method form thread class.* |
| *Best Object Oriented Programming Practice* | *Not a good OOS Practice* |
| *Loosely coupled* | *Tightly coupled* |
| *Improves the reusability of the code* | *Doesn't improve the reusability of the code* |
| *Maintenace of the code will be easy* | *Maintenance of code will be time consuming.* |
| *Separates the task from runner* | *Doesn't separate the task from the runner* |

***FAIL FAST VS FAIL SAFE Iterator in java with exmaples***

|  |  |
| --- | --- |
| ***Failfast*** | ***FailSafe*** |
| *FAIL FAST Iterator doesn’t allow modification of a collection while iterating over it* | *Fail-safe iterator allow modifications of a collection while iterating over it.* |
| *These iterator throw concurrentModificationException if a collection is modified while iterating over it* | *These iterators doesn’t throw any exception, if a collection modifed while iterating over it.* |
| *They use original Collection to traverse over the elements of the collection* | *They use copy of the original collection to travese over the elements of the collection* |
| *These iterator don’t require extra memory* | *Require extra memory* |
| ***ArrayList, Vector, HashMap*** | ***Iterator returned by current HashMap*** |

***Difference between HashSet and HashMap***

|  |  |
| --- | --- |
| ***Hashset*** | ***HashMap*** |
| *HashSet internally uses HashMap* | *HashMap internally uses an array ofn Entry<k,v> objects* |
| *HashSet allows only one null value* | *HashMap allows one null key and multiple null values* |
| *HashSet is slightly slower than HashMap* | *HashMap is Slightly faster than HashSet* |

***Difference between Enumeration and Iterator***

*Enumeration and Iterator two Interfaces of java.util.Package.*

*Enumeration is introduced in JDK 1.0*

*Iterator is introduced in JDK 2.0*

|  |  |
| --- | --- |
| *Enumeration* | *Iterator* |
| *Using Enumeration, you can only traverse the collection. You can’t do any modification to collection while travesing it* | *Using Iterator you can remove an element of the collection while traversing it.* |
| *Enumaration is used to travers the legacy class like vector, stack and HashTabel* | *Iterator is used to iterate most of the class in the collection framewor like arrylist, hashset, hasMap, linked list* |
| *HasMoreElements() and nextElement()* | *HasNext(), next() and remove()* |
| *Enumaration is fail-safe in nature* | *Iterator is fail-fast in nature* |
| *Enumaration is not safe and secure due to its fail fast nature* | *Iterator is safer and secure than Enumeration.* |

|  |  |
| --- | --- |
| ***HASHMAP*** | ***HASHTABLE*** |
| *HashMap is not synchronised and then it is not thread safe* | *HashTable is intrnaly syncronised and threadsfe* |
| *HashMap allows maximum one null key and any number of null values* | *Hashtable doesn’t allow null keys and null values* |
| *Iterator written by HashMap is fail-fast in nature* | *Enumaration is written by HashTable is fail-safe in nature* |
| *HashMap extends AbstractMap* | *HashTable extends Dictonary* |
| *To use HashMap in multi thread applications wrap it sing collection.syncronizedMap()* | *Although HashTable is there to use in multiThread applicatins, now a days it is not preferred, because concurrentHasMap is better option than HashTable.* |

*ConcurrentHashMap*

*Java.util.consurrent.ConsurrentHashMap*

***Difference between Blocked Vs WAITING States in java***

|  |  |
| --- | --- |
| *WAITNG* | *BLOCKED* |
| *A thread will be in this state whien it calls wait() or join() method. The thread will remain in WAITING state until any other threads calls notify() or notifyAll()* | *The thread will be in this state when it is notified by other thread but has not got the object lock yet* |
| *The WAITING thread can be interrupted* | *The BLOCKED thread can’t be interrupted.* |

***Difference between class variables and Instance variables.***

*Method represent behaviour of the object*

*Variables represent state of object variable of two types*

*Class variables*

*Instance variables*

*Class Variables are common to all instances of that class*

*Instance variables are specific to an object.*

*Class variables are also called static variables*

*Class variables are common to all instances of that class I.e these variables will be shared by all the objects of that class, Hence changes made to these variables through one object will reflect in all objects.*

*Class variables can be referred through class name as well as object reference.*

***Illigal forward reference Error in java***

*Illegal forward reference error in java is a compile time error, you will encounter with this error when you try to use a field before it is defined just like in the below example*

***Static import***

*Using static import you can access static members of a class without referring them through their class name.*

*Public class A*

*{*

*public static int I;*

*Public static void methodOne(){ }*

*}*

*The above static members of class A can be directly used in another package if you import them static*

*Package pack2;*

*Import static package1.A.\**

*Pubic class B {*

*Public static void main(String arg[]) {*

*S.o.p(i)*

*}*

*}*

***Marker Interfaces***

*Marker interfaces are interfaces with no members are declared in them.*

*They are just an empty interfaces used to mark or identify a special operation.*

***Ex:*** *Cloneable interface is used to mark clons aperation*

*Serialiable*

*Remote*

***Alternative to Marker Interface***

*I****nternal Flag*** *: can be used to instead of marker interface to indicate any special operation*

***Annotation*** *are recommend instead of marker interface to indicate any special operations.*

***What is a java.lang.class Object***

*Every class in java is associated with one special object type java.lang.class*

*This object hold all the meta data of that class like class name, package name, field, methods annotation details*

*This object is created by JVM when the class is loaded.*

*Event primitive type also have java.lang.class*

*Every Enum, interfaces have their own java.lang.Class.*

***Reflection***

*Reflection is an API which is used to examine or modify the behaviour of a methods, classes , interface at run time.*

*The required class for reflection are provided under java.lang.reflection package*