35+ Concepts Comparison

pramodbablad February 8, 2022 0

1) wait() Vs sleep() In Java

wait()	sleep()
The thread which calls wait() method releases the lock it holds.	The thread which calls sleep() method doesn't release the lock it holds.
The thread regains the lock after other threads call either notify() or notifyAll()	No question of regaining the lock as thread doesn't release the lock.

methods on the same lock.	
wait() method must be called within the synchronized block.	sleep() method can be called within or outside the synchronized block.
wait() method is a member of java.lang.Object class.	sleep() method is a member of java.lang.Thread class.
wait() method is always called on objects.	sleep() method is always called on threads.
wait() is a non-static method of Object class.	sleep() is a static method of Thread class.
Waiting threads can be woken up by other threads by calling notify() or notifyAll() methods.	Sleeping threads can not be woken up by other threads. If done so, thread will throw InterruptedException.
To call wait() method, thread	To call sleep() method, thread need not to

must have object lock.	have object lock.

See More : wait() Vs sleep()

2) Array Vs ArrayList In Java

Array	**ArrayList**
Arrays are static in nature. Arrays are fixed length data structures. You can't change their size once they are created.	ArrayList is dynamic in nature. Its size is automatically increased if you add elements beyond its capacity.
Arrays can hold both primitives as well as objects.	ArrayList can hold only objects.
Arrays can be iterated only through *for* loop or *for-each* loop.	ArrayList provides iterators to iterate through their elements.
The size of an array is checked	The size of an ArrayList can be checked

using *length* attribute.	using *size()* method.
Array gives constant time performance for both add and get operations.	ArrayList also gives constant time performance for both add and get operations provided adding an element doesn't trigger resize.
Arrays don't support generics.	ArrayList supports generics.
Arrays are not type safe.	ArrayList are type safe.
Arrays can be multi-dimensional.	ArrayList can't be multi-dimensional.
Elements are added using assignment operator.	Elements are added using add() method.

See More : Array Vs ArrayList

3) StackOverflowError Vs OutOfMemoryError In Java

StackOverflowError	OutOfMemoryError
It is related to Stack memory.	It is related to heap memory.
It occurs when Stack is full.	It occurs when heap is full.
It is thrown when you call a method and there is no space left in the stack.	It is thrown when you create a new object and there is no space left in the heap.
It occurs when you are calling a method recursively without proper terminating condition.	It occurs when you are creating lots of objects in the heap memory.
How to avoid? Make sure that methods are finishing their execution and leaving the stack memory.	How to avoid? Try to remove references to objects which you don’t need anymore.

See More : StackOverflowError Vs OutOfMemoryError

4) Shallow Copy Vs Deep Copy In Java

Shallow Copy	Deep Copy
Cloned Object and original object are not 100% disjoint.	Cloned Object and original object are 100% disjoint.
Any changes made to cloned object will be reflected in original object or vice versa.	Any changes made to cloned object will not be reflected in original object or vice versa.
Default version of clone method creates the shallow copy of an object.	To create the deep copy of an object, you have to override clone method.
Shallow copy is preferred if an object has only primitive fields.	Deep copy is preferred if an object has references to other objects as fields.
Shallow copy is fast and also less expensive.	Deep copy is slow and very expensive.

See More : Shallow Copy Vs Deep Copy

5) "==" Vs equals() In Java

"==" Operator	**equals() Method**
It is a binary operator in Java.	It is a public method of java.lang.Object class.
It compares the two objects based on their location in the memory.	The default version of equals method also does the comparison of two objects based on their location in the memory. But, you can override the equals method so that it performs the comparison of two objects on some condition.
It can be used on both primitive types	It can be used only on derived types.

as well as on derived types.	
It is best suitable for primitive types.	It is best suitable for derived types.
You can't override the "==" operator. It behaves same for all objects.	You can override the equals method according to your business requirements.

See More : "==" Vs equals()

6) Error Vs Exception In Java

Errors	**Exceptions**
Errors in Java are of type java.lang.Error.	Exceptions in Java are of type java.lang.Exception.
All errors in Java are unchecked type.	Exceptions include both checked as well as unchecked type.
Errors happen	Checked

at run time. They will not be known to compiler.	exceptions are known to compiler where as unchecked exceptions are not known to compiler because they occur at run time.
It is impossible to recover from errors.	You can recover from exceptions by handling them through try-catch blocks.
Errors are mostly caused by the environment in which application is running.	Exceptions are mainly caused by the application itself.
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundExcepti

	on, ClassCastException, NullPointerException

See More : Error Vs Exception

7) Class Variables Vs Instance Variables In Java

Class Variables	**Instance Variables**
Class variables are declared with keyword *static*.	Instance variables are declared without *static* keyword.
Class variables are common to all instances of a class. These variables are shared between the objects of a class.	Instance variables are not shared between the objects of a class. Each instance will have their own copy of instance variables.
As class variables are	As each object will have its

common to all objects of a class, changes made to these variables through one object will reflect in another.	own copy of instance variables, changes made to these variables through one object will not reflect in another object.
Class variables can be accessed using either class name or object reference.	Instance variables can be accessed only through object reference.

See More : Class Variables Vs Instance Variables

8) Fail Fast Vs Fail Safe Iterators In Java

Fail-Fast Iterators	**Fail-Safe Iterators**
Fail-Fast iterators doesn't allow modifications of a collection while iterating over it.	Fail-Safe iterators allow modifications of a collection while iterating over it.

These iterators throw `ConcurrentModificationException` if a collection is modified while iterating over it.	These iterators don't throw any exceptions if a collection is modified while iterating over it.
They use original collection to traverse over the elements of the collection.	They use copy of the original collection to traverse over the elements of the collection.
These iterators don't require extra memory.	These iterators require extra memory to clone the collection.
Ex : Iterators returned by *ArrayList*, *Vector*, *HashMap*.	Ex : Iterator returned by *ConcurrentHashMap.*

See More : Fail-Fast Vs Fail-Safe

9) final Vs finally Vs finalize() In Java

final	finally	finalize()
final is a	finally is	finalize()

keyword in Java which is used to make a variable or a method or a class as unchangeable.	a block in Java which is used for exception handling along with try and catch blocks.	method is a protected method of `java.lang.Object` class which is used to perform some clean up operations on an object before it is removed from the memory.
The value of a variable which is declared as final can't be changed once it is initialized.	finally block is always executed whether an exception is occurred or not and occurred exception	This method is called by garbage collector thread before an object is removed from the memory.

	is handled or not.	
A method declared as final can't be overridden or modified in the sub class and a class declared as final can't be extended.	Most of time, this block is used to close the resources like database connection, I/O resources etc soon after their use.	This method is inherited to every class you create in Java.

See More : final Vs finally Vs finalize

10) ClassNotFoundException Vs NoClassDefFoundError In Java

ClassNotFoundException	NoClassDefFoundError
It is an exception. It is of type java.lang.Exception.	It is an error. It is of type java.lang.Error.

It occurs when an application tries to load a class at run time which is not updated in the classpath.	It occurs when Java runtime system doesn't find a class definition, which is present at compile time, but missing at run time.
It is thrown by the application itself. It is thrown by the methods like Class.forName( ), loadClass() and findSystemClass().	It is thrown by the Java Runtime System.
It occurs when classpath is not updated with required JAR files.	It occurs when required class definition is missing at run time.

See More : ClassNotFoundException Vs NoClassDefFoundError

11) start() Vs run() Methods In Java

start()	run()
New thread is created.	No new thread is created.
Newly created thread executes task kept in run() method.	Calling thread itself executes task kept in run() method.
It is a member of *java.lang.Thread* class.	It is a member of *java.lang.Runnable* interface.
You can't call start() method more than once.	You can call run() method multiple times.
Use of multi-threaded programming concept.	No use of multi-threaded programming concept.

See More : start() Vs run()

12) throw Vs throws Vs Throwable In Java

throw	throws	Throwable
throw is a keyword i	throws is also a	Throwable is a

n Java which is used to throw an exception manually.	keyword in java which is used in the method signature to indicate that this method may throw mentioned exceptions.	super class for all types of errors and exceptions in Java. This class is a member of java.lang package.
Using throw keyword, you can throw an exception from any method or block. But, that exception must be of type **java.lang.Throwable**	The caller to such methods must handle the mentioned exceptions either using try-catch blocks or using throws keyword.	Only instances of this class or it's sub classes are thrown by the java virtual machine or by the throw statement.

class or it's sub classes.	

See More : throw Vs throws Vs Throwable

13) User Threads Vs Daemon Threads In Java

User Threads	**Daemon Threads**
JVM waits for user threads to finish their work. It will not exit until all user threads finish their work.	JVM will not wait for daemon threads to finish their work. It will exit as soon as all user threads finish their work.
User threads are foreground threads.	Daemon threads are background threads.
User threads are high priority threads.	Daemon threads are low priority threads.
User threads are created by	Daemon threads, in

the application.	most of time, are created by the JVM.
User threads are mainly designed to do some specific task.	Daemon threads are designed to support the user threads.
JVM will not force the user threads to terminate. It will wait for user threads to terminate themselves.	JVM will force the daemon threads to terminate if all user threads have finished their work.

See More : User Threads Vs Daemon Threads

14) notify() Vs notifyAll() In Java

notify()	**notifyAll()**
When a thread calls *notify()* method on a particular object, only one thread will be notified which is waiting	When a thread calls *notifyAll()* method on a particular object, all threads which are waiting for the lock of that

for the lock or monitor of that object.	object are notified.
The thread chosen to notify is random i.e randomly one thread will be selected for notification.	All notified threads will get the lock of the object on a priority basis.
Notified thread doesn't get the lock of the object immediately. It gets once the calling thre ad releases the lock of that object. Until that it will be in BLOCKED state. It will move from BLOCKED state to RUNNING state once it gets the lock.	All notified threads will move from WAITING state to BLOCKED state. The thread which gets the lock of the object moves to RUNNING state. The remaining threads will remain in BLOCKED state until they get the object lock.

See More : notify() Vs notifyAll()

15) BLOCKED Vs WAITING States In Java

WAITING	BLOCKED
The thread will be in this state when it calls `wait()` or `join()` method . The thread will remain in WAITING state until any other thread 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 is waiting for notification from other threads.	The BLOCKED thread is waiting for other thread to release the lock.
The WAITING thread can be interrupted.	The BLOCKED thread can't be interrupted.

See More : BLOCKED Vs WAITING

16) Extends Thread Vs Implements Runnable In Java

Implements Runnable	Extends Thread
You can extend any other class.	You can’t extend any other class.
No overhead of additional methods .	Overhead of additional methods from Thread class.
Separates the task from the runner.	Doesn’t separate the task from the runner.
Best object oriented programming p ractice.	Not a good object oriented programming practice.
Loosely coupled.	Tightly coupled.
Improves the reusability of the code.	Doesn’t improve the reusability of the code.
More generalized task.	Thread specific task.
Maintenance of the code will be easy.	Maintenance of the code will be

	time consuming.

See More : Extends Thread Vs Implements Runnable

17) Collection Vs Collections In Java

Collection	Collections
`Collection` is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface.	`Collections` is an utility class in `java.util` package. It consists of only static methods which are used to operate on objects of type Collection.
List, Set and Queue are main sub interfaces of this interface.	Collections.max(), Collections.min(), Collections.sort() are some methods of Collections class.

See More : Collection Vs Collections

18) ArrayList Vs LinkedList In Java

ArrayList	LinkedList
ArrayList is an index based data structure where each element is associated with an index.	Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references

	of previous and next elements are to be changed.
Insertion and removal operations in ArrayList are of order O(n).	Insertion and removal in LinkedList are of order O(1).
Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Retrieval operation in ArrayList is of order of O(1).	Retrieval operation in LinkedList is of order of O(n).
ArrayList is of	LinkedList is

type Random Access. i.e elements can be accessed randomly.	not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it’s index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and

	previous elements.
If your application does more retrieval than the insertions and deletions, then use ArrayList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

See More : ArrayList Vs LinkedList

19) HashMap vs HashSet In Java

HashSet	HashMap
HashSet imple ments Set inter	HashMap imple ments Map inte

face.	rface.
HashSet stores the data as objects.	HashMap stores the data as key-value pairs.
HashSet internally uses HashMap.	HashMap internally uses an array of Entry< K, V> objects.
HashSet doesn't allow duplicate elements.	HashMap doesn't allow duplicate keys, but allows duplicate values.
HashSet allows only one null element.	HashMap allows one null key and multiple null values.
Insertion operation requires only one object.	Insertion operation requires two objects, key and value.
HashSet is slightly slower than HashMap.	HashMap is slightly faster than HashSet.

See More : HashMap Vs HashSet

20) HashMap Vs HashTable In Java

HashMap	HashTable
HashMap is not synchronized and therefore it is not thread safe.	HashTable is internally synchronized and therefore it is thread safe.
HashMap allows maximum one null key and any number of null values.	HashTable doesn’t allow null keys and null values.
Iterators returned by the HashMap are fail-fast in nature.	Enumeration returned by the HashTable are fail-safe in nature.
HashMap extends AbstractMap class.	HashTable extends Dictionary class.
HashMap returns only iterators to traverse.	HashTable returns both Iterator as well as Enumeration for traversal.
HashMap is fast.	HashTable is slow.

HashMap is not a legacy class.	HashTable is a legacy class.
HashMap is preferred in single threaded applications. If you want to use HashMap in multi threaded application, wrap it using Collections.synchronizedMap() method.	Although HashTable is there to use in multi threaded applications, now a days it is not at all preferred. Because, ConcurrentHashMap is better option than HashTable.

See More : HashMap Vs HashTable

21) Iterator Vs ListIterator In Java

Iterator	**ListIterator**
Using Iterator, you can traverse List, Set and Queue type of objects.	But using ListIterator, you can traverse only List objects.
Using Iterator, we can traverse the elements	But, using ListIterator you can traverse

only in forward direction.	the elements in both the directions – forward and backward.
Using Iterator you can only remove the elements from the collection.	But using ListIterator, you can perform modifications (insert, replace, remove) on the list.
You can't iterate a list from the specified index using Iterator.	But using ListIterator, you can iterate a list from the specified index.
Methods : hasNext(), next() and remove()	Methods : hasNext(), hasPrevious(), next(), previous(), nextIndex(), previousIndex(), remove(), set(), add()

See More : Iterator Vs ListIterator

22) ArrayList Vs Vector In Java

ArrayList	Vector
ArrayList is not thread safe.	Vector is thread safe.
As ArrayList is not synchronized, it gives better performance than Vector.	As Vector is synchronized, it is slightly slower than ArrayList.
ArrayList is not a legacy code.	Vector class is considered as legacy, due for deprecation.

See More : ArrayList Vs Vector

23) HashSet Vs TreeSet Vs LinkedHashSet In Java

HashSet	LinkedHashSet	TreeSet
HashSet uses HashMap internally to store it's elements.	LinkedHashSet uses LinkedHashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.

HashSet doesn't maintain any order of elements.	LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted.	TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order.
HashSet gives better performance than the LinkedHashSet and TreeSet.	The performance of LinkedHashSet is between HashSet and TreeSet. It's performance is almost	TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after

	similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements.	each insertion and removal operations.
HashSet gives performance of order O(1) for insertion, removal and retrieval operations.	LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations.	TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations.
HashSet uses	LinkedHashSet	TreeSet uses

equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements.	also uses equals() and hashCode() methods to compare the elements.	compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn't use equals() and hashCode() methods for comparision of elements.
HashSet allows maximum one null element.	LinkedHashSet also allows maximum one null element.	TreeSet doesn't allow even a single null element.

		If you try to insert null element into TreeSet, it throws NullPointerExcepti on.
HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap internally to store its elements.	LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements.	TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap.
Use HashSet if you don’t want to maintain	Use LinkedHashSet if you want to maintain	Use TreeSet if you want to sort the elements

any order of elements.	insertion order of elements.	according to some Comparator.

See More : HashSet Vs LinkedHashSet Vs TreeSet

24) Collections Vs Streams In Java

Collections	Streams
Collections are mainly used to store and group the data.	Streams are mainly used to perform operations on data.
You can add or remove elements from collections.	You can’t add or remove elements from streams.
Collections have to be iterated externally.	Streams are internally iterated.
Collections can be traversed multiple times.	Streams are traversable only once.
Collections are eagerly	Streams are lazily

constructed.	constructed.
Ex : List, Set, Map…	Ex : filtering, mapping, matching…

See More : Collections Vs Streams

25) Java 8 Map() Vs flatMap()

Map()	**flatMap()**
It processes stream of values.	It processes stream of stream of values.
It does only mapping.	It performs mapping as well as flattening.
It's mapper function produces single value for each input value.	It's mapper function produces multiple values for each input value.
It is a One-To-One mapping.	It is a One-To-Many mapping.
Data Transformation	Data Transformation

: From Stream<T> to Stream<R>	: From Stream<Stream <T> to Stream<R>
Use this method when the mapper function is producing a single value for each input value.	Use this method when the mapper function is producing multiple values for each input value.

See More : map() Vs flatMap()

26) Java 8 Stream Intermediate Vs Terminal Operations

Intermediate Operations	**Terminal Operations**
They return stream.	They return non-stream values.
They can be chained together to form a pipeline of operations.	They can’t be chained together.
Pipeline of operations may	Pipeline of operations can

contain any number of intermediate operations.	have maximum one terminal operation, that too at the end.
Intermediate operations are lazily loaded.	Terminal operations are eagerly loaded.
They don't produce end result.	They produce end result.
Examples : filter(), map(), distinct(), sorted(), limit(), skip()	Examples : forEach(), toArray(), reduce(), collect(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()

See More : Intermediate Vs Terminal Operations

27) Iterator Vs Spliterator In Java 8

Iterator	Spliterator

Iterator	Spliterator
It performs only iteration.	It performs splitting as well as iteration.
Iterates elements one by one.	Iterates elements one by one or in bulk.
Most suitable for serial processing.	Most suitable for parallel processing.
Iterates only collection types.	Iterates collections, arrays and streams.
Size is unknown.	You can get exact size or estimate of the size.
Introduced in JDK 1.2.	Introduced in JDK 1.8.
You can’t extract properties of the iterating elements.	You can extract some properties of the iterating elements.
External iteration.	Internal iteration.

See More : Iterator Vs Spliterator

28) Static Binding Vs Dynamic Binding In Java

Static Binding	Dynamic Binding
It is a binding that happens at compile time.	It is a binding that happens at run time.
Actual object is not used for binding.	Actual object is used for binding.
It is also called early binding because binding happens during compilation.	It is also called late binding because binding happens at run time.
Method overloading is the best example of static binding.	Method overriding is the best example of dynamic binding.
Private, static and final methods show static binding.	Other than private, static and final methods show dynamic

Because, they can not be overridden.	binding. Because, they can be overridden.

See More : Static Vs Dynamic Binding

29) Method Overloading Vs Method Overriding In Java

Method Overloading	**Method Overriding**
When a class has more than one method with same name but with different arguments, then we call it as method overloading.	When a super class method is modified in the sub class, then we call this as method overriding.
Overloaded methods must have different method signatures. That means they should differ at least in any one of	Overridden methods must have same method signature. I.e. you must not change the method name, types of

these three things – Number of arguments, Types of arguments and order of arguments. But, they must have same name.	arguments, number of arguments and order of arguments while overriding a super class method.
Overloaded methods can have same or different return types.	The return type of the overridden method must be compatible with that of super class method. That means if super class method has primitive type as its return type, then it must be overridden with same return type. If super class method has derived type as its return type then it must be

	overridden with same type or its sub class type.
Overloaded methods can have same visibility or different visibility.	While overriding a super class method either you can keep the same visibility or you can increase the visibility. But you can’t reduce it.
Overloaded methods can be static or not static. It does not affect the method overloading.	You can’t override a static method.
Binding between method call and method definition happens at compile time (Static Binding).	Binding between method call and method definition happens at run time (Dynamic Binding).

It shows static polymorphism.	It shows dynamic polymorphism.
Private methods can be overloaded.	Private methods can't be overridden.
Final methods can be overloaded.	Final methods can't be overridden.
For method overloading, only one class is required. I.e. Method overloading happens within a class.	For method overriding, two classes are required – super class and sub class. That means method overriding happens between two classes.

See More : Overloading Vs Overriding

30) executeQuery() Vs executeUpdate() Vs execute() In JDBC

executeQuery()	**executeUpdate()**	**execute()**
This	This	This

method is used to execute the SQL statements which retrieve some data from the database.	method is used to execute the SQL statements which update or modify the database.	method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or

		returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML -> INSERT, UPDATE and DELETE DDL -> CREATE, ALTER	This method can be used for any type of SQL statements.

See More : executeQuery() Vs executeUpdate() Vs execute()

31) Statement Vs PreparedStatement Vs CallableStatement In Java

Statement	**PreparedStatement**	**CallableStatement**
It is used to	It is used to	It is used to call the

execute normal SQL queries.	execute parameterized or dynamic SQL queries.	stored procedures.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed.
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
This interface is mainly used for DDL	It is used for any kind of SQL queries w	It is used to execute stored procedur

statements like CREATE, ALTER, DROP etc.	hich are to be executed multiple times.	es and functions.
The performance of this interface is very low.	The performance of this interface is better than the Statement interface (when used for multiple execution of same query).	The performance of this interface is high.

See More : Statement Vs PreparedStatement Vs CallableStatement

32) Process Vs Thread In Java

Process	Thread
Processes are heavy weight	Threads are light weight

operations.	operations.
Every process has its own memory space.	Threads use the memory of the process they belong to.
Inter process communication is slow as processes have different memory address.	Inter thread communication is fast as threads of the same process share the same memory address of the process they belong to.
Context switching between the process is more expensive.	Context switching between threads of the same process is less expensive.
Processes don’t share the memory with other processes.	Threads share the memory with other threads of the same process.

See More : Program Vs Process Vs Threads

33) Checked And Unchecked Exceptions

Checked Exceptions	Unchecked Exceptions
They are known at compile time.	They are known at run time.
They are checked at compile time.	They are not checked at compile time. Because they occur only at run time.
These are compile time exceptions.	These are run time exceptions.
If these exceptions are not handled properly in the application, they give compile time error.	If these exceptions are not handled properly, they don’t give compile time error. But application will be terminated prematurely at run time.
All sub classes of java.lang.Excep	All sub classes of RunTimeExcepti

tion Class except sub classes of RunTimeExcepti on are checked exceptions.	on and sub classes of java.lang.Error are unchecked exceptions.

See More : Checked Vs Unchecked Exceptions

34) HashMap Vs ConcurrentHashMap In Java

HashMap	**ConcurrentHashMap**
HashMap is not synchronized internally and hence it is not thread safe.	ConcurrentHash Map is internally synchronized and hence it is thread safe.
HashMap is the part of Java collection framework since JDK 1.2.	ConcurrentHash Map is introduced in JDK 1.5 as an alternative to HashTable.
HashMap allows maximum one null key and	ConcurrentHash Map doesn’t allow even a single null key and null value.

any number of null values.	
Iterators returned by HashMap are fail-fast in nature.	Iterators returned by ConcurrentHashMap are fail-safe in nature.
HashMap is faster.	ConcurrentHashMap is slower.
Most suitable for single threaded applications.	Most suitable for multi threaded applications.

See More : HashMap Vs ConcurrentHashMap

35) Synchronized HashMap Vs HashTable Vs ConcurrentHashMap In Java

	Synchronized HashMap	**HashTable**	**ConcurrentHashMap**
Locking Level	Object Level	Object Level	Segment Level
Synchronized	All operat	All operati	Only update

operations	ions are synchronized.	ons are synchronized.	operations are synchronized.
How many threads can enter into a map at a time?	Only one thread	Only one thread	By default, 16 threads can perform update operations and any number of threads can perform read operations at a time.
Null Keys And	Allows one null	Doesn’t allow null	Doesn’t allow null

Null Values	key and any number of null values.	keys and null values.	keys and null values.
Nature Of Iterators	Fail-Fast	Fail-Safe	Fail-Safe
Introduced In?	JDK 1.2	JDK 1.1	JDK 1.5
When To Use?	Use only when high level of data consistency is required in multi threaded enviro	Don’t Use. Not recommended as it is a legacy class.	Use in all multi threaded environment except where high level of data consistency is required.

	nment .

See More : Synchronized HashMap Vs HashTable Vs ConcurrentHashMap

36) Servlet Vs GenericServlet Vs HttpServlet In Java

	Servlet	**GenericServlet**	**HttpServlet**
What it is?	Interface	Abstract Class	Abstract Class
Package	javax.servlet	javax.servlet	javax.servlet.http
Hierarchy	Top level interface	Implements Servlet interface	Extends GenericServlet
Methods	init(), service(), destroy(), getServletConfig(),	init(), service(), destroy(), getServletConfig(),	doGet(), doPost(), doPut(), doDelete(),

	getServletInfo()	getServletInfo(), log(), getInitParameter(), getInitParameterNames(), getServletContext(), getServletName()	doHead(), doOptions(), doTrace(), getLastModified(), service()
Abstract Methods	All methods are abstract.	Only service() method is abstract.	No abstract methods.
When to use?	Use it when you want to develop your	Use to write protocol independent	Use to write HTTP-specific servlets.

own Servlet container.	servlets.

See More : Servlet Vs GenericServlet Vs HttpServlet