**Constructor**

* is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes

**getter/setter**

* getter and setter are two conventional methods that are used for retrieving and updating value of a variable. By using getter and setter, the programmer can control how his important variables are accessed and updated in a correct manner, such as changing value of a variable within a specified range.

**Collections:** a single unit of objects. Java Collection framework provides many **interfaces** (Set, List, Queue, Deque) and **classes** (ArrayList, Vector,LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

**Java reference**: class type, interface type, array type, null

**JVM memory area** : Class(Method) Area, Heap, Stack,Program Counter Register, Native Method Stack

**What primitive types are not thread safe?**

* long and double, In Java 64-bit, long and double values were treated as two 32-bit values. That means, a 64-bit write operation is basically performed as two separate 32-bit operations.

**Immutable Class/Object?**

* Objects whose state (the object's data) cannot change after construction. Examples of immutable objects from the JDK include String and Integer.
* Class: once an object is created, we cannot change its content. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable.

**Final, Finalize, Finally**

* Final keyword is used to apply restrictions on class(immutable), method(cannot override) and variable(constant).
* Finally is a block that always executes when the try block exits even if an unexpected exception occurs.
* Finalize is a method to perform clean up processing just before object is garbage collected.

**Access Modifier:**

* Default: within the package. It cannot be accessed from outside the package
* Public
* Protected: within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
* Private: It cannot be accessed from outside the class.

**List, Set, Map**

* List, Set and Map are the interfaces which implements Collection interface.
* Duplicity
* Order: List maintains the insertion order.
* Null value: List allows multiple null values. Set allows 1 null value at most. Map can have 1 null key at most and multiple null values.

**HashMap vs. HashTable**

* HashMap is non-synchronized. It is not thread-safe. HashMap allows one null key and multiple null values.
* HashTable is thread-safe and doesn’t allow any null key or value.
* ConcurrentHashMap: It is thread safe without synchronizing the whole map.There is no locking at the object level. Reads can happen very fast while write is done with a lock.

**How does HashMap work internally?**

* HashMap implements Map interface and maintains key and value pairs.
* HashMap internally works on the principle of Hashing
* HashMap can only contain unique keys and only one null key.
* HashMap methods are non-synchornized.
* HashMap lookups are O(1) in the average case, and O(n) in the worst case

**List, ArrayList, LinkedList**

* ArrayList internally uses a dynamic array to store the elements.

Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. An ArrayList class can act as a list only because it implements List only.

ArrayList is better for storing and accessing data.

* LinkedList internally uses a doubly linked list to store the elements.

Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. LinkedList class can act as a list and queue both because it implements List and Deque interfaces. LinkedList is better for manipulating data.

**What is static?**

* static member is a member of a class that isn’t associated with an instance of a class. It belongs to the class itself. So you can access the static member without first creating a class instance.

**Static Variable**

* It is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc.
* Static variable gets memory only once in the class area at the time of class loading. Using a static variable makes your program more memory efficient (it saves memory). Static variable belongs to the class rather than the object.

**What is pro/con using static in memory management?**

* Pro

1)It has efficient execution time, gets memory only once

2) Static allocation is done at compile time.

* con-

Static members are part of class and thus remain in memory till application terminates and can't be ever garbage collected.

**Abstract class**

* A class that defines attributes and methods for subclasses but is never instantiated.
* Not only can you define a template for children, but Abstract Classes offer the added benefit of letting you define functionality that your child classes can utilize later.
* An abstract class is used if you want to provide a common, implemented functionality among all the implementations of the component. Abstract classes will allow you to partially implement your class, whereas interfaces would have no implementation for any members whatsoever.

**Interface**

* Another way to achieve abstraction in Java
* An interface is a completely "abstract class" that is used to group related methods with empty bodies

**Abstract class VS. Interface**

* Java does not support multiple inheritances but you can implement multiple interfaces in Java.
* Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods.
* Interface has only static and final variables. Abstract class can have final, non-final, static and non-static variables.

**Generic**

* Generic types or methods differ from regular types and methods in that they have type parameters. It is type safe
* Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time

Garbage Collection

* Garbage collection is a process of reclaiming the run-time unused objects.It is performed for memory management. The purpose of garbage collection is to identify and discard those objects that are no longer needed by the application, in order for the resources to be reclaimed and reused.

**Java Bean**

* JavaBean are classes that encapsulate many objects into a single object

**Abstraction**

* Abstraction is a process of hiding the implementation details from the user. Only the functionality will be provided to the user.
* In Java, abstraction is accomplished using Abstract classes and interfaces.

**Main features of OOP**

* Encapsulation

It provides objects with the ability to hide their internal characteristics and behavior.

* **Polymorphism**

Polymorphism is the ability of an object to take on many forms. Common use: when a parent class reference is used to refer to a child class object.

* **Inheritance**

Inheritance is an Object oriented feature which allows a class to inherit behavior and data from other class.

**How to create final class**

* The class must be declared as final (So child classes can't be created).
* Data members in the class must be declared as final (So that we can't change the value of it after object creation)
* A parameterized constructor Getter method for all the variables in it
* No setters(to avoid changing the value of the instance variable)

String

**String** vs. **StringBuffer** vs. **StringBuilder**

* String：immutable and Thread-safe

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.

* StringBuffer: mutable and synchronized/Thread-safe

StringBuffer class doesn't override the equals() method of Object class.

StringBuilder: mutable but **not** synchronized/Thread-safe

* It means 2 threads can call the methods of StringBuilder simultaneously.

**Why stringbuffer not override equals( )?**

* Because StringBuffer is mutable, and its primary use is for constructing strings. If you want to compare content, call StringBuffer#toString() and compare the returned value.

It is not generally useful to override hashCode() for mutable objects, since modifying such an object that is used as a key in a HashMap could cause the stored value to be "lost."

**Why String is immutable?**

* The string is Immutable in Java because String objects are cached in String pool.

**HashCode( ) and equals( )**

* The key point of contract is that if two objects are equal by equals() method then they must have same hashcode, but unequal object can also have same hashcode, which is the cause of collision on hash table based collection e.g HashMap. When you override equals() you must remember to override hashCode() method to keep the contract valid.

**TreeMap**

Based on red black tree, Time Complexity is O(logn), treemap如果放自己写的class <student, Integer>需要comparable或者comparator

Thread

**Process vs. Thread**

* A process is an execution of a program, while a Thread is a single execution sequence within a process. A process can contain multiple threads.
* Processes are independent whereas threads are the subset of process.
* Process have different address space in memory, while threads contain a shared address space.
* Context switching is faster between the threads as compared to processes.
* Inter-process communication is slower and expensive than inter-thread communication.
* Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.

**Multi-threading**

* It is a process of executing multiple threads simultaneously. It is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance.
* Its main advantages are: Threads share the same address space. The cost of communication between the processes is low.

**Ways to create a thread**

* A class may extend the Thread class.
* A class may implement the Runnable interface. The Runnable interface is preferred, as it does not require an object to inherit the Thread class
* An application can use the Executor framework, in order to create a thread pool.

Thread states

* NEW: ready to run, but does not necessarily start running immediately.
* RUNNABLE: JVM is actively executing the thread's code.
* BLOCKED: in a blocked state while waiting for a monitor lock.
* WAITING: waits for another thread to perform a particular action.
* TIMED\_WAITING: The thread waits for another thread to perform a particular action up to a specified waiting time
* TERMINATED: The thread has finished its execution

**Try catch vs. Throw**

* **throws异常之后的代码不会执行，try..catch的会执行**

**Dead Lock**

* A condition that occurs when two processes are waiting for each other to complete, before proceeding. The result is that both processes wait endlessly.

**Wait( ) vs. Sleep( )**

* wait() method is defined in Object class. It releases the lock.
* sleep() method is defined in Thread class. It doesn't release the lock.

Sql

**Normalization**

* Minimizing redundancy and dependency of a database. To add, delete or modify field that can be made in a single table.

**Candidate key**

* It’s also a unique key to identify a record uniquely in a table but a table can have multiple candidate keys. Can null

**Primary Key**

* A primary key is a combination of fields which uniquely specify a row. It’s a special kind of unique key. cannot be NULL/duplicate.

**Composite key**

* A Composite key is a Candidate key or Primary key that consists of more than one attribute.

**Unique Key**

* A Unique key is used to prevent duplicate values in a column. A table can have only one primary key but it can have multiple unique keys

**Foreign Key**

* It is a/more columns in a table whose values correspond to the values of the primary key in another table, accept null/duplicate

**Left Join**

* LEFT JOIN includes all records from the left side and matched rows from the right table, combine column

**Inner Join**

* returns rows when there is a match in both tables. （combine column

**Union**

* It is used to combine the results of two tables, and it eliminates duplicate rows from the tables. (combine row

**Index**

* an index is a pointer to data in a table

Java 8 features

**Lambda expression: (parameter1, parameter2) -> expression**

* It is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

**Stream API**

* a special iterator class that allows us to process collections of objects in a functional manner. It **computes data**, while Collections stores data

**Functional interface**

* an interface with at most one abstract method; implementation can be provided using a Lambda Expression