List of OOPs Concepts in Java

There are four main OOPs concepts in Java. These are:

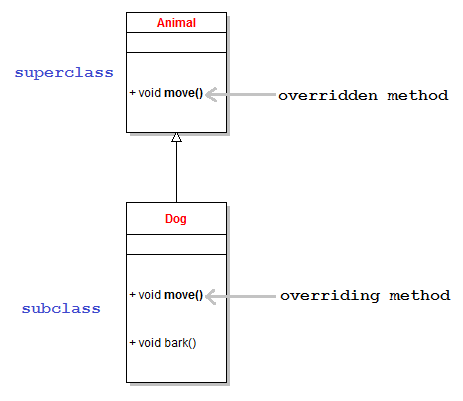
* **Abstraction.** Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don’t need to know how it works in order to enjoy it. In Java, abstraction means simple things like **objects**, **classes**, and **variables** represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.
* **Encapsulation.**This is the practice of keeping fields within a class private, then providing access to them via public methods. It’s a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.
* **Inheritance.**This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.
* **Polymorphism.**This Java OOPs concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. That’s when different meanings are implied by the code itself. The other form is **method overriding**. That’s when the different meanings are implied by the values of the supplied variables. See more on this below.

**Overriding vs. Overloading**

*Overloading* occurs when two or more methods in one class have the same method name but different parameters.

*Overriding* means having two methods with the same method name and parameters (i.e., *method signature*). One of the methods is in the parent class and the other is in the child class. Overriding allows a child class to provide a specific implementation of a method that is already provided its parent class.

**Overriding** refers to the ability of a subclass to re-implement an instance method inherited from a superclass. Let’s take a look at the following class diagram:



**Rule #1:** **Only inherited methods can be overridden.**

**Rule #2:** **Final and static methods cannot be overridden.**

**Rule #3**: **The overriding method must have same argument list.**

**Rule #4**: **The overriding method must have same return type (or subtype).**

**Rule #5**: **The overriding method must not have more restrictive access modifier.**

This rule can be understood as follows:

* If the overridden method is has default access, then the overriding one must be default, protected or public.
* If the overridden method is protected, then the overriding one must be protected or public.
* If the overridden method is public, then the overriding one must be only public.
* **Rule #6**: **The overriding method must not throw new or broader checked exceptions.**
* In other words, the overriding method may throw fewer or narrower checked exceptions, or any unchecked exceptions.

Here are the few other Checked Exceptions –

* SQLException
* IOException
* DataAccessException
* ClassNotFoundException
* InvocationTargetException

###### Static vs final java, both are known **access modifiers** in Java doing different functionalities (used for different jobs).

1. **static:**static keyword can be applied to instance variables and methods but not to classes. When applied, variables and methods can be called without the help of an object. When a method or variable is called without object, encapsulation is not maintained. That is, with static variables and methods, **encapsulation** does not exist.
2. **final:**final keyword can be applied to all constructs – variables, methods and classes. When applied, final behaves very differently with each with different functionalities.

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed. A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue. **java.util.concurrent.Executors** provide implementation of **java.util.concurrent.Executor** interface to create the thread pool in java.  
Let’s write a simple program to explain it’s working.

## yield() method

Theoretically, **to ‘yield’ means to let go, to give up, to surrender**. A yielding thread tells the virtual machine that it’s willing to let other threads be scheduled in its place. This indicates that it’s not doing something too critical. Note that it’s only a hint, though, and not guaranteed to have any effect at all.

## join() method

The join() method of a Thread instance can be **used to “join” the start of a thread’s execution to the end of another thread’s execution** so that a thread will not start running until another thread has ended. If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing.

**sleep()** is a method which is used to hold the process for few seconds or the time you wanted but in case of **wait()** method thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The ***major difference*** is that wait() releases the lock or monitor while sleep() doesn’t releases any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.

HASH MAP

The important points about Java HashMap class are:

* A HashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It maintains no order.

he important points about Java LinkedHashMap class are:

* A LinkedHashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is same as HashMap instead maintains insertion order.

**Comparator vs Comparable**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Comparable** | **Comparator** |
| Sorting logic | Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects | Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc. |
| Implementation | Class whose objects to be sorted must implement this interface.e.g Country class needs to implement comparable to collection of country object by id | Class whose objects to be sorted do not need to implement this interface.Some other class can implement this interface. E.g.-CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id |
| Sorting method | int compareTo(Object o1) This method compares this object with o1 object and returns a integer.Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1 | int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns a integer.Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1 |
| Calling method | Collections.sort(List) Here objects will be sorted on the basis of CompareTo method | Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator |
| Package | Java.lang.Comparable | Java.util.Comparator |

The important points about Java TreeMap class are:

* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

HASH TABLE

* A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.

Here are the few most frequently seen unchecked exceptions –

* NullPointerException
* ArrayIndexOutOfBoundsException
* ArithmeticException
* IllegalArgumentException