# Java Collections interview questions and answers

April 22, 2017 by [T Tak](http://programtalk.com/author/ttak/) [2 Comments](http://programtalk.com/java/java-collections-interview-questions/#comments)

Here is the list of Java Collections interview questions. These questions have been asked over time and are still being asked in interviews. It is good to go through them to understand the various concepts in Collections.

## Java Collections Interview Questions and Answers

### What is a HashSet and how does it work internally in Java?

HashSet and how does it work internally in Java?

*HashSet* is an implementation for the Set interface of Java. It internally uses a *HashMap* to store and retrieve data. It is not a synchronized data structure. So, if multiple threads access and then modify contents of a *HashSet*, then it is the responsibility of the developer to synchronize that part of the code to make sure that the data is not corrupted. If you look at *HashMap* API in oracle documentation, you can see the following:  
public V put(K key, V value)

The map.put returns the original value for the key if present or returns null otherwise.  
public V remove(Object key)

The value V will be the value associated with the key if it was present in the hashmap or null otherwise.  
*HashSet* simply adds an empty *String* to the value part of the *HashMap* when we try to insert an element into it.  
So, if HashSet.add(key) is called, it will internally call map.put(key,””) only if the value returned by the operation is null. Otherwise, it will not add the value to the HashSet.If the HashSet.remove(key) is called, it will remove the value from the map only if the value is present in the map. Based on the success of the above operations, boolean value true or false is returned for HashSet.add(object) and HashSet.remove(Object).  
The implementation of the HashSet can be vaguely written like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import java.io.Serializable;  import java.util.\*;   public class HashSet extends AbstractSet implements Set, Cloneable, Serializable {      private transient HashMap<T, String> map;       public HashSet() {          this(HashMap.DEFAULT\_CAPACITY, HashMap.DEFAULT\_LOAD\_FACTOR);      }       public HashSet(int capacity, int loadFactor) {          this.map = new HashMap<T, String>(capacity, loadFactor);      }       public boolean add(E element) {          return map.put(element, "") == null;      }       public boolean remove(Object element) {          return (map.remove(element) != null);      }      //  further supporting methods for implemented interfaces      //  or abstract methods here...  } |

[Here](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) are the javadocs for HashSet

What is the difference between Collections.synchronizedMap and ConcurrentMap?

Both the classes are thread-safe. The *Collections.synchronizedMap(Map<K,V> map)* returns a map that has a lock over the full object. Only one thread which has acquired the lock over the object can modify the map.  
In the case of *ConcurrentMap*, the map is divided into more than one regions, each of which has a separate lock. So, multiple threads can access/modify the *ConcurrentMap* even though only one thread can do so per region of the Map.

What is the difference between a CheckedMap and an UnmodifiableMap?

*Collections.checkedMap(map)* will return a map that will throw *ClassCastException* if you try to add a wrong type of object into it. It will throw *ClassCastException* if you try to modify existing values also.  
*Collections.unmodifiableMap* is used in case you need to give a read-only view of the data of the map. This is most commonly used in case of DAO’s of various databases where you need to read a set of properties as key-value pairs and present it to the user without letting the user modify them. When you try to add correct type of data or modify existing data, it will throw *UnsupportedOperationException*.

What are fail-fast and fail-safe iterators?

Fail-fast iterators throw a *ConcurrentModificationException* as soon as they see that the user is trying to modify the collection during the iteration process. This is done so that the value of underlying data structure does not get corrupted. An example of this is an Iterato*r* for *ArrayList*.  
Fail-safe iterators do not throw the above-said exception as the iterators iterate over a copy of the actual data structure rather than the original data structure. Example for this is iterator of *CopyOnWriteArrayList*. As the name suggests, the iterator will be iterating over a copy of the list rather than the original one. One drawback of this would be that it is not guaranteed to show updated values to the thread which is manipulating the data structure as it would be working on a copy of the actual data structure.

 How should you design your own class which shall be used as a key in a HashMap?

If you want to design your own class, whose object shall be used as a key in a map, you need to override the *hashCode* and *equals* methods of the superclass Object. They need to follow the *hashcode* and *equals* contract, ie, if *obj1.equals(obj2)* returns true, then *obj1.hashCode()* should be same as *obj2.hashCode()*.

What is a NavigableMap?

*NavigableMap* was introduced in Java1.6. This is a sorted map with navigation capabilities added to it. Since Java 1.6, *TreeMap* implements *NavigableMap*.  
The methods added to *NavigableMap* interface allows you to search through the *Map*. It has methods like *lowerKey()* and *floorKey()* which gives corresponding keys less than and less than or equal to the specified key. *ceilingKey()* and *higherKey()* are the methods which give keys greater than or equal to and greater than the specified key respectively. Similar methods are there to get *Map.Entry* objects as well.  
Another use case of the *NavigableMap* is to get a submap of the given *Map* based on the delimiters specified. The submap thus obtained is actually a backed collection. If we add a value to the original map or the submap which is in the range of the submap, the result is appended to both the maps.  
Let us see that with an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import java.util.\*;    public class TreeMapTest{      public static void main(String[] args){          TreeMap<String,String> treeMap = new TreeMap<String, String>();          treeMap.put("a", "apple");          treeMap.put("c", "cherry");          treeMap.put("p", "pear");          // create a submap here with limits c till n          SortedMap<String, String> subMap = treeMap.subMap("c", "n");          System.out.println("Value of treeMap and subMap before modification...");          System.out.println(treeMap);          System.out.println(subMap);          subMap.put("m", "mango");          System.out.println("Value of treeMap and subMap after modification...");          System.out.println(treeMap);          System.out.println(subMap);          }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Value of treeMap and subMap before modification...  {a=apple, c=cherry, p=pear}  {c=cherry}  Value of treeMap and subMap after modification...  {a=apple, c=cherry, m=mango, p=pear}  {c=cherry, m=mango} |

What is the difference between Iterator and Enumeration?

*Enumeration* and *Iterator* interfaces are found in *java.util* package. Both are used for traversing the collection. But there are subtle differences between both of them which are listed below:

* Using *Iterator*, you can remove an object from the collection while traversing. This is not possible using *Enumeration*. In the case of Enumeration, only traversal is possible.
* *Enumeration* is an interface released during early Java days when collections like *Vector*, *HashTable, etc.,* were predominantly in use. *Iterator* was released later along with other famous members of the collections framework like *ArrayList*, *HashSet*, etc.
* Methods in *Enumeration* are: *hasMoreElements()* and *nextElement()*. Methods in *Iterator* are: *hasNext()*, *next()* and *remove()*.
* Oracle **recommends** using *Iterator* over *Enumeration*.

How can we reverse a List?

*Collections* class contains a static method that allows you to reverse a given list. *Collections.reverse(list)* can be called to achieve this.

What is the difference between HashMap and a HashTable?

Although both *HashMap* and *HashTable* are implementations of the Map interface, there are small differences between them which are as follows:

* *HashMap* is not thread-safe but *HashTable* is threadsafe and synchronized.
* *HashMap* allows one null key and several null values but *HashTable* does not allow any null keys or values.
* *HashMap* can be iterated only using *java.util.Iterator*. *HashTable* provides *Enumeration* as well as *Iterator* for traversal. *Enumeration* is still supported to allow backward compatibility with older versions of Java (like 1.0).
* Since *HashMap* is not synchronized, it will be much faster in non-threaded scenarios than *HashTable*.

What is the difference between LinkedList and ArrayList?

* Both *ArrayList* and *LinkedList* are implementations of the *java.util.List* interface. *LinkedList* implements *java.util.Deque* in addition to the *List* interface. Apart from the methods like *add()* and *remove()* which *LinkedList* shares with *ArrayList*, *addFirst()*, *addLast()*, *removeFirst()* and *removeLast()* methods are also present in *LinkedList* implementation
* *ArrayList* is backed by arrays in underlying implementation. This is a way to realize dynamic arrays (that is, an array with no predetermined size and which grows according to the data added). The underlying implementation of *LinkedList* is a doubly ended *LinkedList*.
* *get(int index)* method returns item at index in constant time in case of an *ArrayList*( as the underlying data structure is an Array). But it takes *O(n)* time in case of a *LinkedList*.
* *Insert(object)* operation is faster in *LinkedList* than that of an ArrayList. This is because each time a new item is added to *ArrayList*, the underlying *Array* backing the list is discarded after copying the values to the newly created *Array* with size (oldarray + 1). This creation of new array and discarding old array is a time-consuming process. But in the case of *LinkedList*, you just need to traverse from head to the particular position and change the previous and next pointers of the list node at the position.
* Memory utilization in case of *LinkedList* is more than that of an array list because of the extra memory required for denoting previous and next nodes of the particular node.

What is a comparator? How different is it from comparable?

*Comparator* and *Comparable* are used when it is required to sort a collection of user defined class or wrapper class’ collection.

**Comparable**

* Implemented by wrapper classes and strings. If you are implementing a custom class, you may use:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class MyCustomClass implements Comparable<MyCustomClass>{        Double weight;        public Double getWeight(){          return this.weight;      }        public MyCustomClass(){}        public MyCustomClass(Double weight){          this.weight = weight;      }        // more members and member functions, //getter, constructors etc go here.      @Override      public int compareTo(MyCustomClass c) {          return this.weight.compareTo(c.getWeight());      }    } |

* Only one way to sort can be done.
* As seen earlier, the class for which the comparison has to be made needs to be changed here.
* Method used is public int compareTo(MyCustomClass c )

**Comparator**

* Implemented by a new class that shall be used alongwith custom class’s collection while calling *Collections.sort()* or *Arrays.sort()*
* Many comparators can be created to sort based on many criterions.
* In this case, you build a separate class which can be used to sort the custom class object’s list.
* Method used is int compare(MyCustomClass c1, MyCustomClass c2);

How would you search for a value in a collection or an array?

In the helper classes like *java.util.Collections* and *java.util.Arrays*, there is a function named *binarySearch* which does this for you. For a given element, *Collections.binarySearch* or *Arrays.binarySearch* will return the position of first occurrence of the element in the collection or array and returns -1 if it is not present in the collection/array.

How to get an array of items from a given List?

You can use *Collections.toArray(list)* to get the contents of the list in form of an array. Similarly, *Arrays.asList(array)* can be used to get the values of the array in list format.

What are concurrent collections?

Java 1.5 introduced *java.util.concurrent* package which contains thread-safe collections that do not throw *ConcurrentModificationException* while trying to modify during a traversal. Some of the significant classes in this package are *CopyOnWriteArrayList*, *ConcurrentHashMap*, *ConcurrentLinkedDeque*, *ArrayBlockingQueue*, etc.

What will be the output for the following code?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import java.util.\*;    public class PriorityQueueTest {        public static void main(String[] args) {          PriorityQueue<String> pQueue = new PriorityQueue<String>();          pQueue.add("Java");          pQueue.add("Collections");          pQueue.add(null);          System.out.println(pQueue.size());      }  } |

The above code will compile properly but during runtime, it would throw a *NullPointerException*. This is because *PriorityQueue* does not allow null values to be added to it.

**What is REST?**

REST stands for Representational State Transfer. It is an architectural style of client-server application

**What are the constraints for REST architectural style?**

REST architectural style describes six contraints:

1. [Uniform Interface](http://programtalk.com/java/restful-api-web-service-interview-questions/#uniforminterface)
2. [Stateless](http://programtalk.com/java/restful-api-web-service-interview-questions/#stateless)
3. [Cacheable](http://programtalk.com/java/restful-api-web-service-interview-questions/#cacheable)
4. [Client-Server](http://programtalk.com/java/restful-api-web-service-interview-questions/#clientserver)
5. [Layered System](http://programtalk.com/java/restful-api-web-service-interview-questions/#layeredsystem)
6. [Code on Demand](http://programtalk.com/java/restful-api-web-service-interview-questions/#codeondemand) – Optional

**What is Uniform Interface?**

Uniform interface constraint defines the interface between clients and servers. It enables each part to develop independently because it simplifies and decouples the architecture. The uniform interface that any REST service must provide is fundamental to its design.

**What are the guiding principles of Uniform Interface?**

 Uniform interface has four guiding principles

**1. Resource Based** – Individual resources are identified in requests using URIs as resource identifiers.  The resources themselves are  separate from the representations that are returned to the client.

**2. Manipulation Of Resources Through Representations** –  When a client holds a representation of a resource including any metadata attached, it has enough information to modify or delete the resource on the server provided it has permission to do so.

**3. Self-descriptive Messages** – Each message includes enough information to describe how to process the message. For example,

**4. Hypermedia as the Engine of Application State (HATEOAS) :**A hypermedia driven site delivers the links contained in the returned body. Here is example of getting Mars planet from HATEOAS service. The service delivers the object and also delivers the link to the object.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | {      "name": "Mars",      "links": [ {          "rel": "self",          "href": "http://localhost:8080/planets/4"      } ]  } |

**What is Stateless?**

The RESTful architecture defines that a server cannot store any of the client state. This  constraint is called Statelessness. So the client has to pass the necessary context with each request. The session state is kept by the client.

**What is Cacheable?**

Responses from servers must define whether a response can be cached or not. This prevents the clients from holding outofdate data or create unwanted requests for data that could be cached.

**What is Client-Server?**

This constraint enforces that the client application and the server application must evolve independently of each other. The Client only should know about the capabilities offered by the a service.

**What is Layered System?**

Between a Client and a server a middleware component should be inserted transparently.  It should be possible to add, modify or reorder the  Layers in case a need arises.

**What is Code On Demand?**

This is an optional constraint. Optional constraint means that architectures that donot support this constraint are still considered as Restful. It allows the clients to download some logical applications like a javascript or flash applicationand execute it on client. Sometimes a client can do some server-side logic on client as they are more efficient in executing this code.

**What us Connectedness?**

Internet as we know it is connected because of the hyperlinks. Similiarly the REST representations are documents with links. As we saw with HATEOS each of the document delivered via rest contains the link to that document.  With links the client gets the guidelines from the server about which states are near the current one.

**What is Idempotence?**

In REST idempotence of an operations means that if a same call is made mulitple times the result of that operation should be same. The number of calls shouldn’t matter. GET, OPTIONS, TRACE and HEAD are idempotent as multiple calls with result in same result. But with DELETE, the first time delete with return with 200 (OK) and will return 404(Not Found) on calls after the first call.

**What is content-negotiation?**

REST services support delivery of more than one version of a document. A representation of a document can be a json, xml, html based on the request. The Client can define the content type requested in the request and the server can respond with that representation of the document.

JAVA RESTful Web Services interview  questions

**What is JAX-RS?**

[JAX-RS API](https://jax-rs-spec.java.net/) defines a set a APIs for the development of Web services built according to the REST architectural style.

**What are some of the commonly used frameworks for Restful webservices in Java?**

There are lots of frameworks out there. Here are some frameworks for

1. [Jersey](https://jersey.java.net/)– Reference implementation of JAX-RS
2. [RESTEasy](http://resteasy.jboss.org/) – JAX-RS implementation
3. [Dropwizard](http://www.dropwizard.io/)
4. [Retrofit](http://square.github.io/retrofit/)
5. [Spark](http://sparkjava.com/)
6. [Spring HATEOAS](http://projects.spring.io/spring-hateoas/) –  can be used with other frameworks

**How can a we secure a RESTful Web Service?**

**Authentication/Authorisation** : It depends on what the aim of the Web service is. And who are the clients of the web service. In general these are some of the ways of securing a RESTful web service. Each of them have some advantages and disadvantages. It all depends on the context.

* Client certificates- Clients have the certificates that are trusted by the servers. The clients presents the trusted certificate on request.
* OAuth with HTTPS
* HTTP Basic with HTTPS
* API Key – API key is provided to client and that API key defines which client is accessing the service

**Securing Aganist Other Threats**

* Secure against SQL injection attacks
* Always POST the sensitive data like password
* Check for the validity of the request. Check for malformed JSON/XML

Tricky Rest API Questions

**If REST applications are supposed to be stateless, how do you manage sessions?**

The REST specification states that no client session state should be stored on server. But the client can manage its own session state and pass that state around so as to get the response from the server. So the server actually is not aware of a client but only for the time it is serving the request from that client.

**Does HTML form support PUT and DELETE methods?**

No, html 5 forms do not support PUT and DELETE methods. But they can be called via javascript using the XmlHttpRequest object.

# Top 15 Core Java Interview Questions

April 2, 2017 by [T Tak](http://programtalk.com/author/ttak/) [Leave a Comment](http://programtalk.com/java/top-corejava-interview-questions/#respond)

Here is a list of core java interview questions.  These are questions that touch the basics of Java language. And it is good to know them even if you are not attending an interview.

### **Q) Describe the contract of equals() and hashCode(). Why is it important that if you implement one, then you must implement both?**

The general contract for the hashCode() and equals() method of the same object is that the hashCode() method should consistently return same value if an object is unchanged. If an object is changed then the hashCode() should return a different value and the changed object should not be equals() to the old object. And for two different objects the contract is that if an object equals() another object then the hashCode() of both the objects should also be same. So it is very important that if one of the methods is overridden then the other one should also be overridden. One important thing to consider.

 It is not required that if two objects are unequal according to the [equals(java.lang.Object)](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

Example Class that overrides both equals() and hashCode()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | package com.programtalk.learn.interview.questions;  /\*\*   \*   \* @author programtalk.com   \*   \*/  public class Person {      private String firstName;        private String lastName;        @Override      public boolean equals(Object obj) {          if (obj == null || !(obj instanceof Person)) {              return false;          }          Person otherPerson = (Person) obj;          return areEqual(firstName, otherPerson.getFirstName()) &amp;&amp; areEqual(lastName, otherPerson.getLastName());      }        private boolean areEqual(String firstString, String secondString) {          if (firstString == secondString) {              return true;          }          return firstString != null ? firstString.equals(secondString) : false;      }        @Override      public int hashCode() {          int hashCode = 17; // start with a prime number          hashCode = firstName != null ? firstName.hashCode(): 0;          hashCode = lastName != null ? lastName.hashCode(): 0;          return hashCode;      }        public String getFirstName() {          return firstName;      }        public void setFirstName(String firstName) {          this.firstName = firstName;      }        public String getLastName() {          return lastName;      }        public void setLastName(String lastName) {          this.lastName = lastName;      }    } |

[Effective Java by Joshua Bloch](http://amzn.to/2kJ9xyD) is great book to read for core java related issues.

Read the java docs of hashCode() [here](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--). You will find some interesting things as to how Object returns distinct integers for distinct objects

Read the java docs of equals() [here](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-). Also look at the Note that refers to the contract defined above.

### **Q) What is the contract between equals() and hashCode() and how it’s used in Java collections?**

We already defined the contract above. Now let’s discuss the importance of this contract in java Collections.

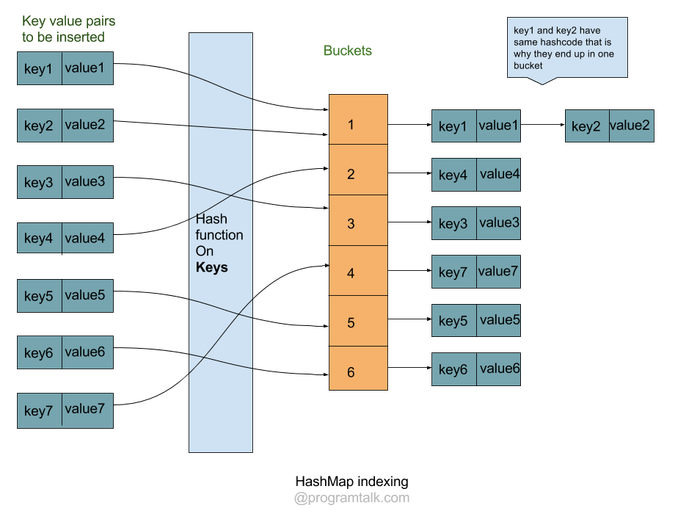
**equals()**

* equals is used in java collections to find an element in a collections.
* remove() methods iterates the list to check if an element in a collection equals the object to be removed
  + Arrays.asList(“111”, “222”).remove(“111”)
* contains() method checks if a method is in collections. It also iterates the list and checks using equals whether an element in the list is equal to the object to be checked against.
  + Arrays.asList("111", "222").contains("1111")

**hashCode()**

When an object is inserted into HashMap, HashTable or HashSet, a hashcode is generated and used to store the objects. To insert an entry to HashMap,  a key and a value is needed. When a key is inserted to a HashMap, the hashCode() and equals() method of the key are used to store the key-value in a bucket( Bucket means each element of the HashMap). If a hashCode() generates the same hashcode for two keys that have equals as false, then both key-value pairs are stored in one bucket as a linked list.  And when a search is done on such a key, the linkedlist has to be iterated to find the key that equals the searched one .

Let’s take the worst case possible;  if all the unequal keys have same hashcode, then they would be stored in one bucket as a linkedlist in the HashMap. And hence the HashMap would practically become a LinkedList and search would be very very slow. So it is very important that the keys of the HashMap should have different hashcode so that they are stored in different buckets in HashMap. HashMap performance is the best when there is approximately one key in one bucket.  So it is very important to implement the hashCode() method properly and even effort should be made that unequal objects have not the same hashcode

[](https://i0.wp.com/programtalk.com/wp-content/uploads/2017/02/HashMapIndexing.png)

HashMap

### **Q) List three Collections interfaces and the basic contract of each. List concrete implementations of each, how they differ, and performance characteristics in space and time.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **List<E>** | **Set<E>** | **Map<K,V>** |
| ***Implementations*** | LinkedList ArrayList | HashSet LinkedHashSet TreeSet | IdentityHashMap HashMap TreeMap |
| ***Duplicates*** | Allowed | Not Allowed | Duplicate keys not allowed |
| ***Order of Elements*** | List is ordered collection. The List interface by itself enforces the order on the implementation | A Set can be an ordered Collection. But the Set interface doesn’t enforce it. | Only a key-value pair can be inserted. |

You can find the complete answer to this question at [differences of List, Set and Map](http://programtalk.com/java/difference-list-set-map/).

### **Q) Describe Java collections, interfaces, implementations. What’s the difference between LinkedList and ArrayList?**

The difference between LinkedList and ArrayList is best explained [here on stackoverflow.](http://stackoverflow.com/questions/322715/when-to-use-linkedlist-over-arraylist)

### **Q) What’s the difference between primitives and wrappers, where do they live, what’s autoboxing?**

Primitives types are the basic data types in java. And the wrapper classes are classes that encapsulates these primitive types. Here is the list of primitive types and the corresponding wrapper classes in Java.

|  |  |
| --- | --- |
| Primitive Type | Wrapper Class |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

**Differences between primitive and Wrapper classes**

1. Declaration
   * for primitive types, variables are declared e.g. int i = 0;
   * for wrapper types, a wrapper class has to be instantiated e.g; Integer i = new Integer(1);
2. Assignment of null value
   * primitives don’t have a null value. (e,g. when declaring a variable, that usually cannot have a negative value is initialized with -1 as to know that the value comes from a declaration rather than computation)
   * wrapper classes can be initialized will null e.g. Integer i = null;. But this makes the program vulnerable to NullPointerException([a billion dollar mistake](https://en.wikipedia.org/wiki/Tony_Hoare#Apologies_and_retractions))
3. Collections
   * primitives cannot be inserted into collections
   * only wrapper types can be inserted into collections.
4. Memory
   * primitives takes less memory
   * Wrapper classes have the primitive values and in addition have the other overheads that lead to more memory consumption

Some other things that need to be considered here are  autoboxing . We will discuss it in next question.

Next thing that you should know is that BigDecimal and BigInteger are wrapper classes  but are not primitive wrapper classes. These are immutable wrapper classes

### **Q) What is autoboxing?**

In java a an int type can be assigned directly to the Integer wrapper class.

|  |  |
| --- | --- |
| 1  2 | int i = 1;  Integer iWrapper = i; |

Behind this assignment Java creates an Integer object from the int type. That is called autoboxing.  I have written and article about autoboxing and it also gives the various things that should be known to a developer, please read [Autoboxing](http://programtalk.com/java/autoboxing-and-autounboxing/).

### **Q) Where can you put keywords final and static and what do they do?**

**final**keyword is used to define a primitive variable as immutable. Once a variable has been declared final, it is not allowed to assign anything to that variable. The variable can no longer be changed. Why i did not say that it makes the variable immutable? Because the objects can be changed by the methods exposed by an object so the objects are still mutable. But the primitive variables cannot be changed except by assigning them to some value. So by defining them as final they become immutable. A final variable can be initialized on declaration or within a constructor. A final variable that is not initialized on declaration is called blank final. Anonymous inner classes declared within a method can only access the final variables available in the method scope.

**final**keyword can be used with a method and a class. A **final method** can not be overriding by a subclass. A final class can’t be inherited by any class. **java.lang.String**is a final class.

**static** keyword on a variable declares that variable belongs to a class rather than instance of that class. So only one instance of static field exists.  static keyword on a method means that the methods belongs to a class. A **static method** is available without creating an instance of class and can be invoked by ClassName.staticMethod(). Static methods cannot access the fields of the class unless the fields are marked as static. Fields can be accessed via an instance of the class. static keyword can also be used with an import to import a static method or a static field of another class.

### **Q) Access modifiers in Java.**

Access modifiers are used to determine who can access a class, method or a member of a class. There are two levels of access control.

1. Top Level class :  A top level class can have  either
   1. public : accessible to anyone.
   2. package private(no modifier): accessible only to the package level classes
2. Members of class : At member level there are four types of access modifiers. A class has access to all its members independent of the modifier.
   1. public: accessible to everyone
   2. protected : accessible to the package and the subclasses
   3. package private(no modifier): only accessible within the same package as the class
   4. private : only accessible within the class and noone else.

### **Q) Difference between String and StringBuilder / StringBuffer.**

To represent a string literals like “Hello” in java a String class is used. The String class is a sequence of characters. The String class is immutable. I have disussed  [why string is immutable?](http://programtalk.com/java/string-immutable-java/) The String class provides various method to concatenate, substring or creating a copy of the String. All these methods do not modify the String but they all return a new String. A String represents a string in UTF-16 format.

**StringBuilder** is a mutable object and can be used for creating strings. A StringBuilder has append method to create a string and append any String to the existing String. To change a String object, you need to create a new one. StringBuilder can append to strings without creating new objects and hence has a better performance. See my post for [String vs StringBuilder vs stringBuffer performance](http://programtalk.com/java/runtime-comparison-string-concatenation/).

**StringBuffer** has exactly the same properties as Stringbuilder but the only differnce between the two is that StringBuffer is safe to use in mutithreaded environment as StringBuffer is synchronized. So it is advised to use StringBuilder in a single threaded environment so to have a performance benefit over StringBuffer.

### **Q) Difference between interface and abstract class.**

interface is a contract. Till java 8 no implementation was allowed in interfaces but with java 8 default methods interfaces can also have implementation in methods.In Interface all the methods take the same access modifier as declared for the interface. Any class implementing a interface has to implement all the methods from the interface except the default methods. A class can implement more than one interface.

Abstract class is a class that can have abstract methods that have no implementation and can also have methods that have implementation.  Only the method declared have to be implemented by the extending class. Since java doesn’t allow multiple inheritance of classes, a class can only extend one abstract class.

### **Q) Difference between overriding and overloading.**

You can see a detailed answer here in [java oops concept polymorphism](http://programtalk.com/java/oops-interview-questions-answers/#polymorphism)

### **Q) Types of exceptions and “handle or declare” rule.**

Java has three kinds of exception:

**1. Checked Exceptions**: Checked exceptions are the excpetions that need to be either handled with catch or throw. Checked Exceptions are checked at compile time. Even a checked Exception is not handled, the code will not compile. In the below example we will read a file and if the file is not found the readFile() method will throw IOException and in the main method we simply print the stacktrace.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package com.programtalk.learn.java.beginner;    import java.io.File;  import java.io.FileReader;  import java.io.IOException;    public class ReadFile {      public static void main(String[] args) {            try {              System.out.println(readFile());          } catch (IOException e) {              e.printStackTrace();          }      }        private static String readFile() throws IOException {          String content = null;          File file2Read = new File("myfile.txt");          FileReader reader = null;          try {              reader = new FileReader(file2Read);              char[] chars = new char[(int) file2Read.length()];              reader.read(chars);              content = new String(chars);              reader.close();          } finally {              if (reader != null) {                  reader.close();              }          }          return content;      }  } |

**2. Error**: Errors are exceptional conditions that are external to an application. Java does not enforce the handling of Errors. So if a method throws an Error, the calling method doesn’t need to handle it. The code will compile fine. An application cannot predict an error condition on compile time. Suppose if an application opens a file and cannot read a file due to some issue with the hardware, throwing java.io.IOError.

**3. Runtime Exceptions**: These exceptions are internal to an application. Usually these exceptions are because of some bugs with the application code. NullPointerException is a famous example of runtime exception. The below code has a bug that make the code throw a NullPointerException.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public static void main(String[] args) {      printIfHarry(null);  }    public static void printIfHarry(String firstName){      if(firstName.equals("Harri")){          System.out.println("I am Harray");      }  } |

So there is a bug in the program and we need to fix it by either having a null check before invoking equals as below. So we don’t need to handle the exception but fix the bug.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static void printIfHarry(String firstName){      if(firstName != null && firstName.equals("Harri")){          System.out.println("I am Harray");      }  } |

Handle or Declare:  If you are calling a method that is throwing an exception, a decision needs to be made whether exception has to be caught or thrown to the caller and make the caller handle the exception. The rule is to bubble up the exception to the place where it can handled best. Like a

### **Q) How does garbage collector work?**

Object data is stored in heap. Heap is managed by garbage collector. A garbage collector can be configured at startup. Automatic garbage collection is a process of managing the heap memory. The garbage collector looks for the objects that any not used anymore. It does so by looking at the references of the object. If an object is not referenced from anywhere in the application, the objects are garbage collected and the memory used by these objects are reclaimed.

[Here](http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html) is a article that explains how garbage collector works for Oracle implementation of Java. One the nice book for performance you can read [here](http://amzn.to/2lVC2ZO).

### **Q) How to make a class immutable and what’s the point?**

A class can be made immutable by declaring all the variables in the class as final. So that the state of the class cannot be changed. The immutability of any object makes that object safe for use as the user would be sure that the state of the object can’t change while the object is being used. The best example of immutability is the String class. An immutable object doesn’t need synchronized access on fields as the same value would be avaliable to all the threads and none of the threads can change the value of the object. Let’s make a Person class immutable.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package com.programtalk.learn.java.beginner;  /\*\*   \*   \* @author programtalk.com   \*   \*/  public class Person {        private final String firstName;      private final String lastName;        public Person(String firstName, String lastName) {          this.firstName = firstName;          this.lastName = lastName;      }        public String getFirstName() {          return firstName;      }        public String getLastName() {          return lastName;      }    } |

The state of a Person Object cannot be updated. It can only accept the firstName and lastName on creation and then nothing can be changed.

**Q) What’s JIT compilation?**

You can find the complete answer to this question at [JIT compiler](http://programtalk.com/java/jit-compiler/).

### **Q) What’s new in Java 8 / Java 7? What’s coming in Java 9?**

In [new features of Java 8  with examples](http://programtalk.com/java/java-8-new-features/) I have discussed various new features. Here is the list of features:

1. Lambda expressions
2. Functional interfaces
3. Default and static methods in interfaces
4. forEach() in Iterable interface
5. Stream API
6. Java Date/Time API
7. Concurrency improvements

In [New Features in Java 9](http://programtalk.com/java/java-9-new-features/), I have discussed all the major features that are coming in Java 9. Here is the list of major changes:

1. Module System (Jigsaw Project)
2. JShell
3. Collection Factory methods
4. Private Methods in Interfaces
5. Reactive Streams
6. Multi Resolution Images API
7. Process API
8. Try-With Resources
9. Diamond Operator Extension
10. Deprated annotation Enhanced
11. Unified JVM Logging
12. SafeVarargs Scope Extension
13. HTTP 2 Client
14. HTML5 Javadoc

### **Q) What is a  default method?**

In Java 8, a major change was introduced in interfaces. The interfaces can now have implementation of methods if they are defined with keyword as default. I have written a detailed article about it at [Default methods in Java 8](http://programtalk.com/java/java8-static-default-methods-interface/).

You may also be interested in:

* [Java OOPs concepts interview questions](http://programtalk.com/java/oops-interview-questions-answers/)
* [RESTful Web Service interview questions](http://programtalk.com/java/restful-api-web-service-interview-questions/)
* [Check Palindrome String in Java](http://programtalk.com/java/check-palindrome-string-java/)
* [Increasing subsequences – Leetcode](http://programtalk.com/java/increasing-subsequences-leetcode/)
* [Largest Palindrome Product – Leetcode](http://programtalk.com/java/largest-palindrome-product-leetcode/)
* [calculate the Hamming distance for two given integers](http://programtalk.com/java/hamming-distance-two-integers/)
* [Binary Tree Paths – Given a binary tree, return all root-to-leaf paths](http://programtalk.com/java/binary-tree-paths-given-binary-tree-return-root-leaf-paths/)
* [Why is String immutable in Java](http://programtalk.com/java/string-immutable-java/)

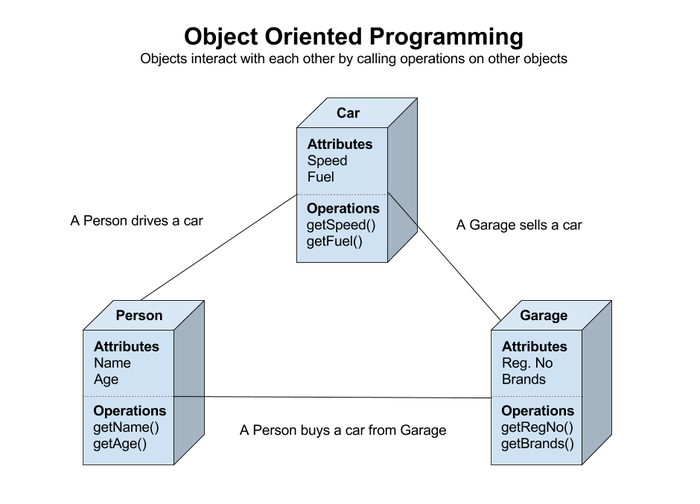
# Java OOPs interview questions and answers

February 4, 2017 by [T Tak](http://programtalk.com/author/ttak/) [Leave a Comment](http://programtalk.com/java/oops-interview-questions-answers/#respond)

It is very important to know the concepts of Object Oriented Programming(OOP) before you go for an interview for any of the OOPs languages like Java, C++ or Python.  Here I would be listing the mostly asked OOPs interview questions. And would also explain various OOPs concepts.

**1. What is Object Oriented Programming?**

Object Oriented programming is a style of programming that is based on the concept of objects. Objects advertise the type of data that it will store and the types of operations that it allows to manipulate the data.

[](https://i2.wp.com/programtalk.com/wp-content/uploads/2017/02/Object-oriented-programming.png)

Object oriented programming

**2. What are the core concepts of OOPS?**

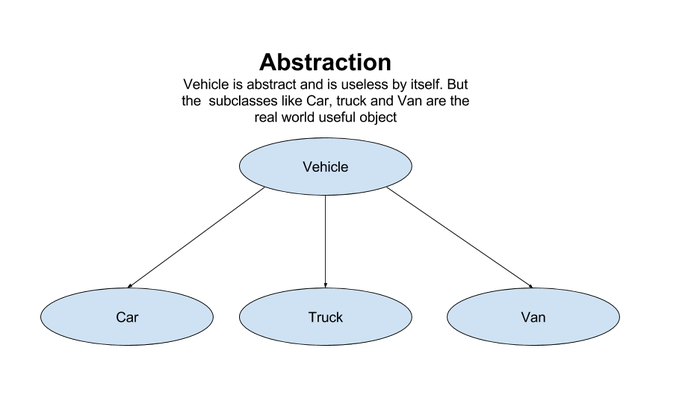
The different OOPS concepts are

* [Abstraction](http://programtalk.com/java/oops-interview-questions-answers/#abstraction)
* [Encapsulation](http://programtalk.com/java/oops-interview-questions-answers/#encapsulation)
* [Inheritance](http://programtalk.com/java/oops-interview-questions-answers/#inheritance)
* P[olymorphism](http://programtalk.com/java/oops-interview-questions-answers/#polymorphism)
* [Composition](http://programtalk.com/java/oops-interview-questions-answers/#composition)
* [Association](http://programtalk.com/java/oops-interview-questions-answers/#association)
* [Aggregation](http://programtalk.com/java/oops-interview-questions-answers/#aggregation)
* [Dependency](http://programtalk.com/java/oops-interview-questions-answers/#dependency)

**3. What is Abstraction?**

Abstraction is a the concept that denotes the extracting of essential details about an item or a group of items while ignoring the inessential details. In Java abstraction can be achieved by defining an Abstract class and extending that class or by defining an interface.

|  |  |
| --- | --- |
| 1  2  3  4 | public interface Vehcile {    public String getModel();    public String getType();  } |

[](https://i1.wp.com/programtalk.com/wp-content/uploads/2017/02/abstraction-interview.png)

abstraction

**4. What is Encapsulation?**

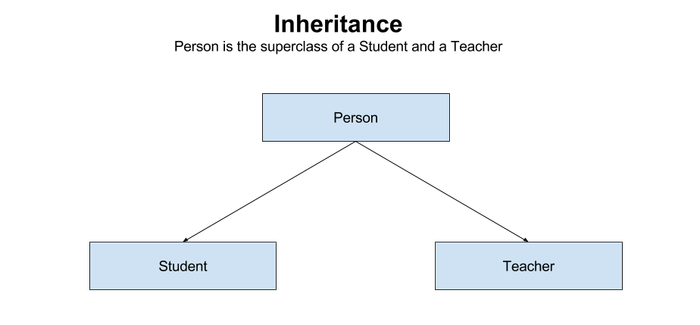
Encapsulation is grouping of all closely related data into a Class. The class contains all the data and methods. The attributes are not exposed to the outside world by defining them private and the methods are exposed by defining them as public. Essentially it hides the details of how things work and only making the behaviour public. Here is a typical java bean that hides the attributes but exposes the get and set method on the attributes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package com.programtalk.learn.encapsulation;    public class Car {      private String model;      private String type;        public String getModel() {          return model;      }      public void setModel(String model) {          this.model = model;      }      public String getType() {          return type;      }      public void setType(String type) {          this.type = type;      }  } |

**5. What is Inheritance?**

Inheritance is a mechanism by which one class is derived from another class. In Java, classes can inherit the properties and methods of other classes. The class that is derived from another class is called a subclass and the class that is inherited is called a superclass. Here is nice example of  inheritance. A Student is a subclass of Person. So the student can use the properties and methods of Person and doesn’t need to define them again.

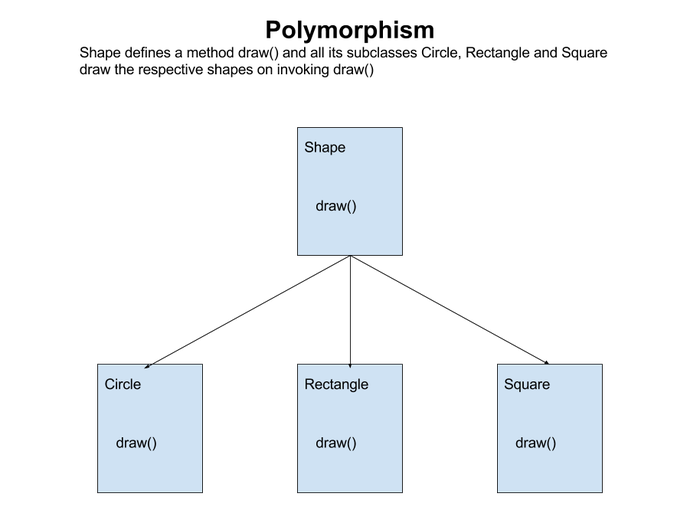
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.programtalk.learn.inheritance;    public class Person {      private String name;      private String age;        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public String getAge() {          return age;      }        public void setAge(String age) {          this.age = age;      }  } |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package com.programtalk.learn.inheritance;  public class Student extends Person{      private String grade;        public String getGrade() {          return grade;      }        public void setGrade(String grade) {          this.grade = grade;      }  } |

[](https://i1.wp.com/programtalk.com/wp-content/uploads/2017/02/Inheritance.png)

Inheritance

**6. What is Polymorphism?**

Polymorphism is a concept that more than one type of objects share a common interface but have a different functionality. Example is that of various shapes.  Calling the draw method on any of shapes draws the respective shape.

[](https://i2.wp.com/programtalk.com/wp-content/uploads/2017/02/ploymorphism.png)

ploymorphism

All Java classes inherit class Object and hence all java objects are polymorphic. Concepts that demonstrate polymorphism in java:

**Method Overloading**:

In Java same method name can be given to more than one methods but the arguments or parameters have to be different Here is a simple example. A class Operations defines three methods with same name add but with different argument list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | package com.programtalk.learn.methodoverloading;    public class Operations{        private int add(int i, int j) {          return i + j;      }      private long add(long i, long j) {          return i + j;      }      private float add(float i, float j) {          return i + j;      }  } |

**Method Overriding:**

In Method overriding a subclass method changes the behavior of the SuperClass method by redefining the method and overriding its functionality. Here is a nice example of method overriding.

* the example calls draw() on shape first and it prints “I am a shape”
* then it calls draw() on circle and it prints “I am a shape”
* then it creates an instance of Circle but assigns it to Shape. This is possible because Circle is also a type of Shape as it extends Shape. Now when draw() is invoked here. It prints “I am a Circle”.  This behavior is referred to as **virtual method invocation.** An overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | package com.programtalk.learn.methodoverriding;    class Shape {        public void draw(){          System.out.println("I am a shape");      }  }    class Circle extends Shape {        public void draw(){          System.out.println("I am a circle");      }  }    public class TestOverriding{        public static void main(String[] args) {          Circle circle = new Circle();          circle.draw();            Shape shape = new Shape();          shape.draw();          // this is the interesting case          Shape circleShape = new Circle();          circleShape.draw();        }  } |

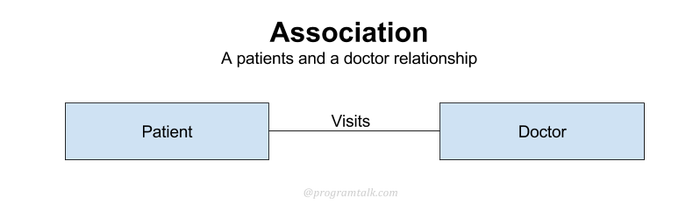
Output

|  |  |
| --- | --- |
| 1  2  3 | I am a circle  I am a shape  I am a circle |

You can also find more details [here](https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html)

**7. What is Association?**

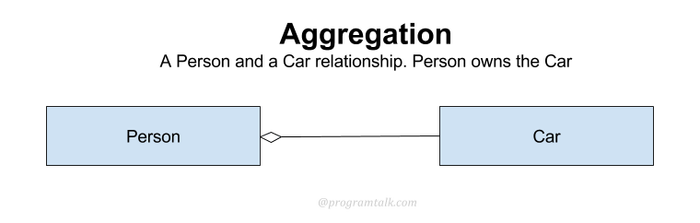
It represents a relationship between two or more objects where all objects have their own lifecycle and there is no owner. Let us take the example of Doctor and a Patient. Both have their own lifecycle. A doctor can see multiple patients and a patient can visit multiple doctors.

[](https://i2.wp.com/programtalk.com/wp-content/uploads/2017/02/Association.png)

Association

**8. What is Aggregation?**

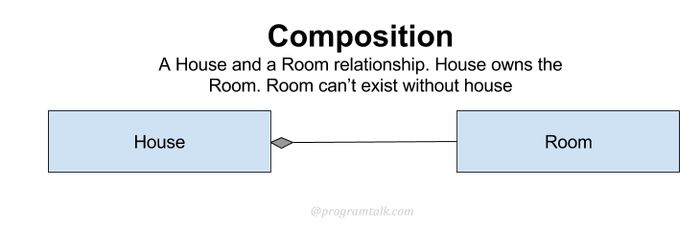
Aggregation is a form of association with all objects having there own lifecycle but there is an ownership. An example here would be like a car belongs to a Person but if the Person object is deleted the car object would not be deleted as it has its own lifecycle.

[](https://i1.wp.com/programtalk.com/wp-content/uploads/2017/02/Aggregation-1.png)

Aggregation

**9. What is Composition?**

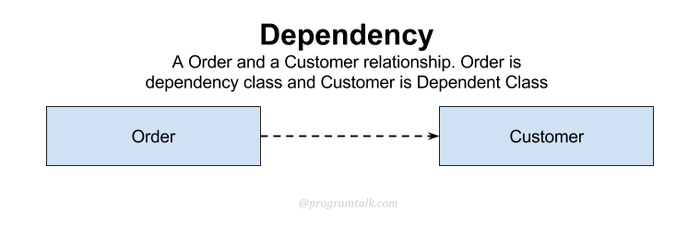
Composition is a special form of Aggregation. If the owner is destroyed then the owned object is also destroyed. The child object doesn’t have its own lifecycle

[](https://i0.wp.com/programtalk.com/wp-content/uploads/2017/02/Composition.png)

Composition

**10. What is Dependency?**

Dependency is a a form of relationship in which one object depends on another object. The other may or may not depend on the first object.  An Order object depends on Customer object. An Order Object cannot be created without the Customer object.

[](https://i1.wp.com/programtalk.com/wp-content/uploads/2017/02/Dependency.png)

Dependency

**11. What is multiple inheritance? Does Java support multiple inheritance?**

Multiple inheritance is a feature in which an object can inherit attributes and methods of the parent object. Let’s look at various types of multiple inheritance and which of them is supported by Java.

**Multiple Inheritance of State:**This means ability to inherit fields from multiple classes. Java does not permit to extend more than once class. So this type of multiple inheritance is not supported in Java.

**Multiple inheritance of implementation:**This means the ability to inherit method definitions from multiple classes. Java didn’t support this type of multiple inheritance untile Java 8. In Java 8 [Default methods](https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html) were introduced. A class can implement more than one interface. And these interfaces can have default methods with same signature. The Java compiler provides some rules to determine which default method a particular class uses.

**Multiple inheritance of type:** Ability of a class to implement more than one interface. Java supports this type of multiple inheritance.

**12. Is Java a pure Object Oriented Programming language?**

Java is not a pure Object Oriented language.  Java does have support for encapsulation, Abstraction, Inheritance, Polymorphism. But Java fails with a key requirement for pure OOP language which is that every predefined type must be Object. But java has primitive types like int,long, char, float.  In pure OOP language all the operations have to be via Object methods but Java allows arithmetic operations like String a= "hello" + " world!"

**ArrayList vs HashSet Java**

Main difference between ArrayList and HashSet is that one is a List implementation while other is a Set implementation. It means all the differences between a List data structure and a Set data structure also applies to this pair. For example, List implementations are ordered, it store element in the order they were added, while Set implementation doesn't provide such guarantee. Similarly, since List provides Random access, you can access any element directly if you know the index, but Set doesn't provide such facility. You need to Iterate through whole collection to get access of any elements. We will see couple of more difference in this Java tutorial. By the way ArrayList and HashSet are two most common Collection class used in Java programming language and before discussing difference between ArrayList vs HashSet, let's see some similarities between them

## Difference between ArrayList vs HashSet in Java

Here are couple of differences between ArrayList and HashSet in Java:

1) First and most important difference between ArrayList and HashSet is that ArrayList implements List interface while HashSet implements Set interface in Java.

2) Another difference between ArrayList and HashSet is that [ArrayListallow duplicates](http://java67.blogspot.sg/2015/03/how-to-remove-duplicates-from-arraylist.html) while HashSet doesn't allow duplicates. This is the side effect of fist difference and property of implementing List and Set interface.

3) Third difference between ArrayList and HashSet is that ArrayList is an ordered collection and maintains insertion order of elements while HashSet is an unordered collection and doesn't maintain any order.

4) Fourth difference between ArrayList and HashSet is that ArrayList is backed by an Array while HashSet is backed by an HashMap instance. See [how HashSet internally works in Java](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html) for more details.

5) Fifth difference between HashSet and ArrayList is that its index based you can retrieve object by calling get(index) or remove objects by calling remove(index) while HashSet is completely object based. HashSet also doesn't provide get() method.

That's all on difference between ArrayList and HashSet. these differences helps you to decide where to use ArrayList and where to use HashSet in Java. in terms of performance between ArrayList and HashSet, choose what suits best to you. raw array is fasted among them.

Read more: <http://www.java67.com/2012/07/difference-between-arraylist-hashset-in-java.html#ixzz4i0NcFaZa>

### What is the difference between Process and Thread?

A process is a self contained execution environment and it can be seen as a program or application whereas Thread is a single task of execution within the process. Java runtime environment runs as a single process which contains different classes and programs as processes. Thread can be called lightweight process. Thread requires less resources to create and exists in the process, thread shares the process resources.

### What are the benefits of multi-threaded programming?

In Multi-Threaded programming, multiple threads are executing concurrently that improves the performance because CPU is not idle incase some thread is waiting to get some resources. Multiple threads share the heap memory, so it’s good to create multiple threads to execute some task rather than creating multiple processes. For example, Servlets are better in performance than CGI because Servlet support multi-threading but CGI doesn’t.

### What is difference between user Thread and daemon Thread?

When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

### How can we create a Thread in Java?

There are two ways to create Thread in Java – first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class. Read this post to learn more about [creating threads in java](http://www.journaldev.com/1016/java-thread-example).

### What are different states in lifecycle of Thread?

When we create a Thread in java program, its state is New. Then we start the thread that change it’s state to Runnable. Thread Scheduler is responsible to allocate CPU to threads in Runnable thread pool and change their state to Running. Other Thread states are Waiting, Blocked and Dead. Read this post to learn more about [life cycle of thread](http://www.journaldev.com/1044/thread-life-cycle-in-java-thread-states-in-java).

### Can we call run() method of a Thread class?

Yes, we can call run() method of a Thread class but then it will behave like a normal method. To actually execute it in a Thread, we need to start it using **Thread.start()** method.

### How can we pause the execution of a Thread for specific time?

We can use Thread class sleep() method to pause the execution of Thread for certain time. Note that this will not stop the processing of thread for specific time, once the thread awake from sleep, it’s state gets changed to runnable and based on thread scheduling, it gets executed.

### What do you understand about Thread Priority?

Every thread has a priority, usually higher priority thread gets precedence in execution but it depends on Thread Scheduler implementation that is OS dependent. We can specify the priority of thread but it doesn’t guarantee that higher priority thread will get executed before lower priority thread. Thread priority is an int whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

### What is Thread Scheduler and Time Slicing?

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads. Once we create and start a thread, it’s execution depends on the implementation of Thread Scheduler. Time Slicing is the process to divide the available CPU time to the available runnable threads. Allocation of CPU time to threads can be based on thread priority or the thread waiting for longer time will get more priority in getting CPU time. Thread scheduling can’t be controlled by java, so it’s always better to control it from application itself.

### What is context-switching in multi-threading?

Context Switching is the process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time. Context Switching is the essential feature for multitasking operating system and support for multi-threaded environment.

### How can we make sure main() is the last thread to finish in Java Program?

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function. Here is an article about [Thread join method](http://www.journaldev.com/1024/java-thread-join-example).

### How does thread communicate with each other?

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource. Check this post to learn more about [thread wait, notify and notifyAll](http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example).

### Why thread communication methods wait(), notify() and notifyAll() are in Object class?

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

### Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

### Why Thread sleep() and yield() methods are static?

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

### How can we achieve thread safety in Java?

There are several ways to achieve thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes. Learn more at [thread safety tutorial](http://www.journaldev.com/1061/thread-safety-in-java).

### What is volatile keyword in Java

When we use volatile keyword with a variable, all the threads read it’s value directly from the memory and don’t cache it. This makes sure that the value read is the same as in the memory.

### Which is more preferred – Synchronized method or Synchronized block?

Synchronized block is more preferred way because it doesn’t lock the Object, synchronized methods lock the Object and if there are multiple synchronization blocks in the class, even though they are not related, it will stop them from execution and put them in wait state to get the lock on Object.

### How to create daemon thread in Java?

Thread class setDaemon(true) can be used to create daemon thread in java. We need to call this method before calling start() method else it will throw IllegalThreadStateException.

### What is ThreadLocal?

Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it’s variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables.  
Every thread has it’s own ThreadLocal variable and they can use it’s get() and set() methods to get the default value or change it’s value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread. Check this post for small example program showing [ThreadLocal Example](http://www.journaldev.com/1076/java-threadlocal-example).

### What is Thread Group? Why it’s advised not to use it?

ThreadGroup is a class which was intended to provide information about a thread group. ThreadGroup API is weak and it doesn’t have any functionality that is not provided by Thread. Two of the major feature it had are to get the list of active threads in a thread group and to set the uncaught exception handler for the thread. But Java 1.5 has added setUncaughtExceptionHandler(UncaughtExceptionHandler eh) method using which we can add uncaught exception handler to the thread. So ThreadGroup is obsolete and hence not advised to use anymore.

t1.setUncaughtExceptionHandler(new UncaughtExceptionHandler(){

@Override

public void uncaughtException(Thread t, Throwable e) {

System.out.println("exception occured:"+e.getMessage());

}

});

### What is Java Thread Dump, How can we get Java Thread dump of a Program?

Thread dump is list of all the threads active in the JVM, thread dumps are very helpful in analyzing bottlenecks in the application and analyzing deadlock situations. There are many ways using which we can generate Thread dump – Using Profiler, Kill -3 command, jstack tool etc. I prefer jstack tool to generate thread dump of a program because it’s easy to use and comes with JDK installation. Since it’s a terminal based tool, we can create script to generate thread dump at regular intervals to analyze it later on. Read this post to know more about [generating thread dump in java](http://www.journaldev.com/1053/java-thread-dump-visualvm-jstack-kill-3-jcmd).

### What is Deadlock? How to analyze and avoid deadlock situation?

Deadlock is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

To analyze a deadlock, we need to look at the java thread dump of the application, we need to look out for the threads with state as BLOCKED and then the resources it’s waiting to lock, every resource has a unique ID using which we can find which thread is already holding the lock on the object.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation, read this post to learn how to [analyze deadlock in java](http://www.journaldev.com/1058/deadlock-in-java-example) with sample program.

### What is Java Timer Class? How to schedule a task to run after specific interval?

java.util.Timer is a utility class that can be used to schedule a thread to be executed at certain time in future. Java Timer class can be used to schedule a task to be run one-time or to be run at regular intervals.

java.util.TimerTask is an [**abstract class**](http://www.journaldev.com/1582/abstract-class-in-java) that implements Runnable interface and we need to extend this class to create our own TimerTask that can be scheduled using java Timer class.

Check this post for [java Timer example](http://www.journaldev.com/1050/java-timer-timertask-example).

### What is Thread Pool? How can we create Thread Pool in Java?

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. [Thread Pool Example](http://www.journaldev.com/1069/threadpoolexecutor-java-thread-pool-example-executorservice) program shows how to create and use Thread Pool in java. Or read [ScheduledThreadPoolExecutor Example](http://www.journaldev.com/2340/java-scheduler-scheduledexecutorservice-scheduledthreadpoolexecutor-example) to know how to schedule tasks after certain delay.

### What will happen if we don’t override Thread class run() method?

Thread class run() method code is as shown below.

public void run() {

if (target != null) {

target.run();

}

}

Above target set in the init() method of Thread class and if we create an instance of Thread class as new TestThread(), it’s set to null. So nothing will happen if we don’t override the run() method. Below is a simple example demonstrating this.

public class TestThread extends Thread {

//not overriding Thread.run() method

//main method, can be in other class too

public static void main(String args[]){

Thread t = new TestThread();

System.out.println("Before starting thread");

t.start();

System.out.println("After starting thread");

}

}

It will print only below output and terminate.

Before starting thread

After starting thread

## Java Concurrency Interview Questions and Answers

### What is atomic operation? What are atomic classes in Java Concurrency API?

Atomic operations are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.

int++ is not an atomic operation. So by the time one threads read it’s value and increment it by one, other thread has read the older value leading to wrong result.

To solve this issue, we will have to make sure that increment operation on count is atomic, we can do that using Synchronization but Java 5 java.util.concurrent.atomic provides wrapper classes for int and long that can be used to achieve this atomically without usage of Synchronization. Go to this article to learn more about [atomic concurrent classes](http://www.journaldev.com/1095/atomicinteger-java).

### What is Lock interface in Java Concurrency API? What are it’s benefits over synchronization?

Lock interface provide more extensive locking operations than can be obtained using synchronized methods and statements. They allow more flexible structuring, may have quite different properties, and may support multiple associated Condition objects.  
The advantages of a lock are

* + it’s possible to make them fair
  + it’s possible to make a thread responsive to interruption while waiting on a Lock object.
  + it’s possible to try to acquire the lock, but return immediately or after a timeout if the lock can’t be acquired
  + it’s possible to acquire and release locks in different scopes, and in different orders

Read more at [**Java Lock Example**](http://www.journaldev.com/2377/java-lock-example-reentrantlock).

### What is Executors Framework?

In Java 5, Executor framework was introduced with the java.util.concurrent.Executor interface.

The Executor framework is a framework for standardizing invocation, scheduling, execution, and control of asynchronous tasks according to a set of execution policies.

Creating a lot many threads with no bounds to the maximum threshold can cause application to run out of heap memory. So, creating a ThreadPool is a better solution as a finite number of threads can be pooled and reused. Executors framework facilitate process of creating Thread pools in java. Check out this post to learn with example code to [create thread pool using Executors framework](http://www.journaldev.com/1069/threadpoolexecutor-java-thread-pool-example-executorservice).

### What is BlockingQueue? How can we implement Producer-Consumer problem using Blocking Queue?

java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue doesn’t accept null values and throw NullPointerException if you try to store null value in the queue.

BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.

BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem.  
Check this post for [producer-consumer problem implementation using BlockingQueue](http://www.journaldev.com/1034/java-blockingqueue-example).

### What is Callable and Future?

Java 5 introduced java.util.concurrent.Callable interface in concurrency package that is similar to Runnable interface but it can return any Object and able to throw Exception.

Callable interface use Generic to define the return type of Object. Executors class provide useful methods to execute Callable in a thread pool. Since callable tasks run in parallel, we have to wait for the returned Object. Callable tasks return java.util.concurrent.Future object. Using Future we can find out the status of the Callable task and get the returned Object. It provides get() method that can wait for the Callable to finish and then return the result.  
Check this post for [Callable Future Example](http://www.journaldev.com/1090/java-callable-future-example).

### What is FutureTask Class?

FutureTask is the base implementation class of Future interface and we can use it with Executors for asynchronous processing. Most of the time we don’t need to use FutureTask class but it comes real handy if we want to override some of the methods of Future interface and want to keep most of the base implementation. We can just extend this class and override the methods according to our requirements. Check out [**Java FutureTask Example**](http://www.journaldev.com/1650/java-futuretask-example-program) post to learn how to use it and what are different methods it has.

### What are Concurrent Collection Classes?

Java Collection classes are fail-fast which means that if the Collection will be changed while some thread is traversing over it using iterator, the iterator.next() will throw ConcurrentModificationException.

Concurrent Collection classes support full concurrency of retrievals and adjustable expected concurrency for updates.  
Major classes are ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet, check this post to learn [how to avoid ConcurrentModificationException when using iterator](http://www.journaldev.com/378/java-util-concurrentmodificationexception).

### What is Executors Class?

Executors class provide utility methods for Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes.

Executors class can be used to easily create Thread Pool in java, also this is the only class supporting execution of Callable implementations.

### What are some of the improvements in Concurrency API in Java 8?

Some important concurrent API enhancements are:

* + ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.
  + CompletableFuture that may be explicitly completed (setting its value and status).
  + Executors newWorkStealingPool() method to create a work-stealing thread pool using all available processors as its target parallelism level.

**Recommended Read**: [Java 8 Features](http://www.journaldev.com/2389/java-8-features-with-examples)

That’s all for Java Thread and Concurrency interview questions, I have been adding more to this list. So bookmark the post for future reference.

Filed Under: [Interview Questions](http://www.journaldev.com/dev/interview-questions), [Java](http://www.journaldev.com/dev/java)

ExecutorService executorService = Executors.newFixedThreadPool(10);

executorService.execute(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

executorService.shutdown();

## ExecutorService Example

Here is a simple Java ExectorService example:

ExecutorService executorService = Executors.newFixedThreadPool(10);

executorService.execute(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

executorService.shutdown();

First an ExecutorService is created using the newFixedThreadPool() factory method. This creates a thread pool with 10 threads executing tasks.

Second, an anonymous implementation of the Runnable interface is passed to the execute() method. This causes the Runnable to be executed by one of the threads in the ExecutorService.

## Task Delegation

Here is a diagram illustrating a thread delegating a task to an ExecutorService for asynchronous execution:

|  |
| --- |
| A thread delegating a task to an ExecutorService for asynchronous execution. |
| **A thread delegating a task to an ExecutorService for asynchronous execution.** |

Once the thread has delegated the task to the ExecutorService, the thread continues its own execution independent of the execution of that task.

## ExecutorService Implementations

Since ExecutorService is an interface, you need to its implementations in order to make any use of it. The ExecutorService has the following implementation in the java.util.concurrent package:

* [ThreadPoolExecutor](http://tutorials.jenkov.com/java-util-concurrent/threadpoolexecutor.html)
* [ScheduledThreadPoolExecutor](http://tutorials.jenkov.com/java-util-concurrent/scheduledexecutorservice.html)

## Creating an ExecutorService

How you create an ExecutorService depends on the implementation you use. However, you can use the Executors factory class to create ExecutorService instances too. Here are a few examples of creating an ExecutorService:

ExecutorService executorService1 = Executors.newSingleThreadExecutor();

ExecutorService executorService2 = Executors.newFixedThreadPool(10);

ExecutorService executorService3 = Executors.newScheduledThreadPool(10);

## ExecutorService Usage

There are a few different ways to delegate tasks for execution to an ExecutorService:

* execute(Runnable)
* submit(Runnable)
* submit(Callable)
* invokeAny(...)
* invokeAll(...)

I will take a look at each of these methods in the following sections.

### execute(Runnable)

The execute(Runnable) method takes a java.lang.Runnable object, and executes it asynchronously. Here is an example of executing a Runnable with an ExecutorService:

ExecutorService executorService = Executors.newSingleThreadExecutor();

executorService.execute(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

executorService.shutdown();

There is no way of obtaining the result of the executed Runnable, if necessary. You will have to use a Callable for that (explained in the following sections).

### submit(Runnable)

The submit(Runnable) method also takes a Runnable implementation, but returns a Future object. This Future object can be used to check if the Runnable as finished executing.

Here is a ExecutorService submit() example:

Future future = executorService.submit(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

future.get(); //returns null if the task has finished correctly.

### submit(Callable)

The submit(Callable) method is similar to the submit(Runnable) method except for the type of parameter it takes. The Callable instance is very similar to a Runnable except that its call() method can return a result. The Runnable.run() method cannot return a result.

The Callable's result can be obtained via the Future object returned by the submit(Callable) method. Here is an ExecutorService Callable example:

Future future = executorService.submit(new Callable(){

public Object call() throws Exception {

System.out.println("Asynchronous Callable");

return "Callable Result";

}

});

System.out.println("future.get() = " + future.get());

The above code example will output this:

Asynchronous Callable

future.get() = Callable Result

### invokeAny()

The invokeAny() method takes a collection of Callable objects, or subinterfaces of Callable. Invoking this method does not return a Future, but returns the result of one of the Callable objects. You have no guarantee about which of the Callable's results you get. Just one of the ones that finish.

If one of the tasks complete (or throws an exception), the rest of the Callable's are cancelled.

Here is a code example:

ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<Callable<String>>();

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 1";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 2";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 3";

}

});

String result = executorService.invokeAny(callables);

System.out.println("result = " + result);

executorService.shutdown();

This code example will print out the object returned by one of the Callable's in the given collection. I have tried running it a few times, and the result changes. Sometimes it is "Task 1", sometimes "Task 2" etc.

### invokeAll()

The invokeAll() method invokes all of the Callable objects you pass to it in the collection passed as parameter. The invokeAll() returns a list of Future objects via which you can obtain the results of the executions of each Callable.

Keep in mind that a task might finish due to an exception, so it may not have "succeeded". There is no way on a Future to tell the difference.

Here is a code example:

ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<Callable<String>>();

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 1";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 2";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 3";

}

});

List<Future<String>> futures = executorService.invokeAll(callables);

for(Future<String> future : futures){

System.out.println("future.get = " + future.get());

}

executorService.shutdown();

## ExecutorService Shutdown

When you are done using the ExecutorService you should shut it down, so the threads do not keep running.

For instance, if your application is started via a main() method and your main thread exits your application, the application will keep running if you have an active ExexutorService in your application. The active threads inside this ExecutorService prevents the JVM from shutting down.

To terminate the threads inside the ExecutorService you call its shutdown() method. The ExecutorService will not shut down immediately, but it will no longer accept new tasks, and once all threads have finished current tasks, the ExecutorService shuts down. All tasks submitted to the ExecutorService before shutdown() is called, are executed.

If you want to shut down the ExecutorService immediately, you can call the shutdownNow() method. This will attempt to stop all executing tasks right away, and skips all submitted but non-processed tasks. There are no guarantees given about the executing tasks. Perhaps they stop, perhaps the execute until the end. It is a best effort attempt.

## 18. Using the @SpringBootApplication annotation

Many Spring Boot developers always have their main class annotated with @Configuration, @EnableAutoConfiguration and @ComponentScan. Since these annotations are so frequently used together (especially if you follow the [best practices](http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-structuring-your-code.html) above), Spring Boot provides a convenient @SpringBootApplication alternative.

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes:

package com.example.myproject;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication* // same as @Configuration @EnableAutoConfiguration @ComponentScan

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**GITHUB**

---github account:  
<https://github.com/join>  
<https://github.com>  
username: arnulfonunez  
email: [arnulfonunez@hotmail.com](mailto:arnulfonunez@hotmail.com)  
password: wksumz00

<https://github.com/arnulfonunez/ionic2-course.git>  
git remote add origin <https://github.com/arnulfonunez/ionic2-course.git>  
git push -u origin master

<https://github.com/arnulfonunez/ionic2-recipe.git/>

<https://ionicframework.com/getting-started/>

Lessons: 7, 10,11,12

Visual Studio Code

<https://developer.android.com/studio/index.html#downloads>

<http://stackoverflow.com/questions/37505709/how-do-i-download-the-android-sdk-without-downloading-an-android-studio>

[https://github.com/driftyco/ionic](https://github.com/driftyco/ionic/blob/2.0/CHANGELOG.md#200-beta8-2016-06-06)

Steps:

--install node.js

--install Cordova (link between app and phone)

--install Ionic 2

--install angular

--install Typescript

--install google chrome browser

--install ios 'sdk' (if building on mac)

- npm install -g cordova

- npm install -g ionic

- npm install -g typescript

search for "cordova android" to find the android jdk recommended for the cordova version that

we use. Check version: cordova platform ls

ionic id:

Arnulfo Nunez

arnulfonunez

[arnulfonunez@hotmail.com](mailto:arnulfonunez@hotmail.com)

wksumz00

Create blank project: ionic start myFirstProject blank

start project: ionic serve

create code for android: ionic platform add android

create code for ios: ionic platform add ios

start on platform specific: ionic serve --lab

ionic info

npm uninstall -g cordova

npm uninstall -g typescript

npm cache clean -f

npm install npm -g

npm install -g ionic@beta

**From:** Nunez, Arnulfo (IT - New York Park Ave)   
**Sent:** Monday, March 20, 2017 11:31 AM  
**To:** Nunez, Arnulfo (IT - New York Park Ave)  
**Subject:** Git Tutorials

<https://www.atlassian.com/git/tutorials/setting-up-a-repository>

git config --global user.name "John Smith"

git config --global user.email [john@example.com](mailto:john@example.com)

git add: tells git that you want to include updates to a particular file in the next commit

git add <file>:  stage all changes in <file> for the next commit

git add <directory>. Stage all changes in <directory> for the next commit

git add –p: begin an interative staging session tha lets you choose portions of a file to add to the next commit.

git add .: stage all changes for the next commit

Developing a project revolves around the basic edit/stage/commit pattern.

Git commit: commit files and ask for message

Git commit –m “message” : commit files and use the provided message

Git commit –a: commit files and make sure that only files that at some point have been added with git add will be commited.

Git stash: git stash temporally shelves or stashes changes you have made to your working copy so you can work on something else.

Please note that stash are local and they are not transferred to the server when you push.

Git stash save “message”: creates stash and gives a meaningful message

Git stash pop: reapply previously stashed changes with git stash. Popping your stash removes the changes from your stash and reapplies them to your working copy.

Git stash pop stash@{1}: pop the stash at position 1

Git stash apply: reapply previously stashed changes with git stash. Apply does not remove the changes from your stash.

Git stash –u: include untracked files in the stash

Git stash list: list the stashes created

Git revert: git revert command undoes a committed snapshot.

Git commit –amend: amending commit replaces it the commit. Do not use –amend if I have push the changes to the origin version as it may affect other users.

Git rebase: rebasing is the process of moving a branch to a new base commit.

Git checkout –b newBranch existingBranch: this creates a branch newBranch based on existingBranch and checkout the new created branch.

Git branch branchName: create a new branch branchName based on the current checked out branch

Git reflog: shows updates done to the branches

Git merge master branchName: merge master into branchName

git commit -a

git push origin charXXXX

git checkout PARENT

git pull origin PARENT

git merge charXXXX or git merge charXXXX PARENT: merge branch into PARENT

fix any conflicts

git commit -a

git push origin PARENT

<https://github.com/arnulfonunez/ionic2-recipe.git/>

My.visualstudio.com

personal

[arnulfo.nunez@stifel.com](mailto:arnulfo.nunez@stifel.com)

wksumz00A

Plural: [arnulfo.nunez@stifel.com](mailto:arnulfo.nunez@stifel.com)

wksumz00A

<https://coursetro.com/posts/code/55/How-to-Install-an-Angular-4-App>

**CREATE NODE JS SERVER TO TEST ANGULAR**

* 1. Create folder : mkdir server\_name
  2. Go to new folder: cd server\_name
  3. Create new nodejs server: npm init
  4. Select all the default values
  5. Install formidable: npm install –save formidable (this will help us process forms.
  6. Code for the node:

var http = require('http');

var formidable = require('formidable');

var util = require('util');

var server = http.createServer(function(req, res) {

res.setHeader('Access-Control-Allow-Origin','\*');

res.setHeader("Access-Control-Allow-Headers", "Access-Control-Allow-Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers");

var reqMethod = req.method.toLowerCase();

if (reqMethod == 'post') {

processForm(req,res);

return;

}

if(reqMethod == 'get')

{

var data = {

data:{

languages:['English','Spanish','German','Other']

}

};

var responseData = JSON.stringify(data);

res.end(responseData);

console.log("get: ",responseData);

}

res.end();

});

function processForm(req,res){

var form = new formidable.IncomingForm();

form.parse(req,function(err,fields){

res.writeHead(200,{'content-type':'text/plain'});

var data = JSON.stringify({data:fields});

res.end(data);

console.log('posted fields:\n');

console.log(data);

});

};

var port = 3100;

server.listen(port);

console.log("Server listening on port " + port);

Run the server: node server.js

Install bootstrap components like date picker and time picker

Npm install ng2-bootstrap –save

<http://valor-software.com/ngx-bootstrap/#/>

**Lambda Example**

**public** List<TransactionBean> getAltList()

{

List<TransactionBean> retVal = **new** ArrayList<TransactionBean>();

**try**

{

retVal = jdbcTemplate.query(***ALT\_LIST\_SQL***, **new** BeanPropertyRowMapper(TransactionBean.**class**));

retVal = jdbcTemplate.query(***ALT\_LIST\_SQL***,

(ResultSet param, **int** paramInt) -> {

TransactionBean b = **new** TransactionBean();

b.setTransactionDescription("this is my test");

**return** b;

}

);

Comparator<TransactionBean> tt = (TransactionBean t1, TransactionBean t2)->t1.getAmount().compareTo(t2.getAmount()); //Create comparator with regular lambda

retVal.sort(tt);

retVal.sort((TransactionBean t1, TransactionBean t2) -> t1.getAmount().compareTo(t2.getAmount()) ); //Regular lambda

retVal.sort((t1, t2) -> t1.getAmount().compareTo(t2.getAmount())); //Use type inferance

Comparator<TransactionBean> tComparator = Comparator.*comparing*((TransactionBean t1) -> t1.getAmount()); //Use comparator

retVal.sort(tComparator);

retVal.sort( *comparing*((bean) -> bean.getAmount()) ); //Use comparing with type infered

retVal.sort( *comparing*(TransactionBean::getAmount) ); //User method reference

retVal.sort( *comparing*(TransactionBean::getAmount).reversed());

Predicate<TransactionBean> predicate = (t) -> t.getAmount() > 0;

/\*

public abstract interface RowMapper<T>

{

public abstract T mapRow(ResultSet paramResultSet, int paramInt)

throws SQLException;

}

\*/

}

**catch**(Exception e){

e.printStackTrace();

}

**return** retVal;

}