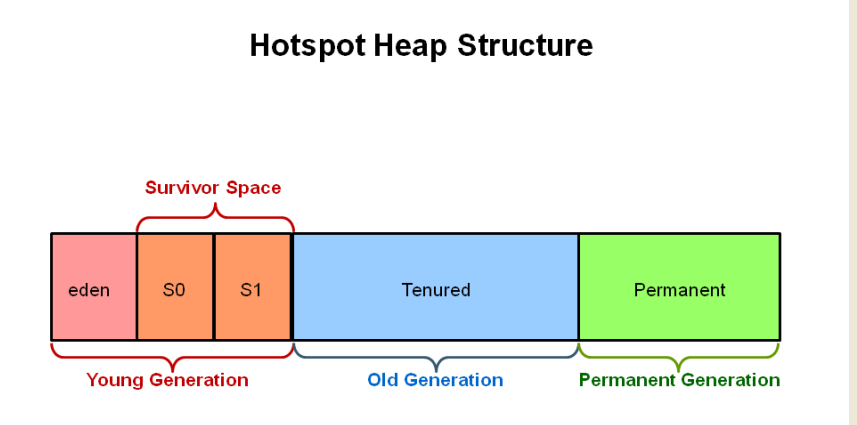
* **Java key interview questions**

**Anagram program**

1. **JVM Memory Model?**

****

**Young Generation**

In Eden space new objects are created. When the space is filled up to a certain percentage, GC is performed. This event is called **as Minor GC**.

The surviving objects from Eden space are moved to survivor spaces(S0 & S1).

Minor GC checks the object in survivor spaces and move to other survivor spaces (S0->S1).

Objects surviving multiple minor GC are moved to old generation.

**Old Generation**

When an old generation is full, the major GC takes place to remove the unused objects. This event can cause a minor pause in the working of application. Too many frequent major GC can trigger performance issue in the application. While designing an application, one should be considerate about it.

**Perm Gen**

Perm Gen contains the metadata of the classes.

1. Names of the classes
2. Methods of a class
3. Constant pool(e.g String pool) information.
4. Internal objects created by the JVM
5. Object arrays and type arrays associated with a class (e.g., an object array containing references to methods).

Note: Perm Gen is not available in Java 8 anymore

When an object does becomes eligible for GC.

When a object is not referred by other objects or all its references are set to null.

Object moves out of the scope.

Weak reference objects, such as WeakHashMap

1. **Can we execute a program without main () method?**

Yes, one of the ways is static block.

1. **What is static variable?**

Static variable gets memory only once in class area at the time of class loading.

Static variable is used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

1. **What is static method? imp**

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

Static method can access static data member and can change the value of it.

1. **Is it possible to instantiate the abstract class? IMP**

No, abstract class can never be instantiated.

1. **Can Abstract class have constructor? IMP**

Yes abstract class can have constructor. But still we cannot instantiate the abstract class. The constructor of abstract class can be used to initialize the instance variables of abstract class.

1. **Static binding and Dynamic binding**? **IMP**

First of all let me tell you what is binding.

Association of method definition to the method call is known as binding.

There are 2 types of binding

**Static Binding / Early Binding**

Binding resolved at compile time by compiler is known as static or early binding.

All the private, static and final methods have always been bonded at compile-time.

**Why binding of Static, final and private methods is always a static binding?**

Compiler knows that all such methods cannot be overridden and will always be accessed by object of local class.

**Dynamic Binding or Late Binding**

When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late binding.

Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method. Thus while calling the overridden method, the compiler gets confused between parent and child class method (since both the methods have same name).

Static Binding vs Dynamic Binding

1. **Different ways to create java object? IMP**
2. **Using new keyword:** Using this we can call whichever constructor we want to call no arg constructor or parameterized constructor

Employee e = new Employee();

Employee e1 = new Employee(name);

1. **Using newInstance() method of Class class**: This method calls no-arg constructor to create object.

**Employee e = (Employee) Class**.forName(“com.huawei.Employee”)

**Or**

Employee e = Employee.class.newInstance();

1. **Using newInstance() method of constructor class:**

We can also call parameterized constructor or private constructor

Constructor<Employee> constructor = Employee

Employee emp = constructor.newInstance();

1. **Using a clone() method**:

Employee emp4 = (Employee) emp3.clone();

1. **Using Deserialization:**

ObjectInputStream in = new ObjectInputStream()

Employee emp5 = (Employee) in.readObject();

Search question with dzone key to check example and more details about this feature

1. **Difference between Exception and Error? IMP**

**Exceptions:** can be caught and dealt by the application

Ex: IOException.

**Error:** are the scenarios that are out of the scope of the application.It is not possible to anticipate them

**Ex:** hardware failure, JVM crash or out of memory error

1. **Say super class method does not declare an exception can subclass methods declare exception. If super class method declares an exception can subclass method declare exception?**
2. **What is difference between final, finally and finalize in Java?**

final and finally are keywords in java whereas finalize is a method.

**final** keyword can be used with class variables so that they can’t be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses.

**finally** keyword is used with try-catch block to provide statements that will always gets executed irrespective of whether exception arises or not, usually finally is used to close resource.

**finalize()** method is executed by Garbage Collector before the object is destroyed, it’s great way to make sure all the global resources are closed.

1. **How to create immutable class in java? IMP**

To create immutable class in java, you have to do following steps.

1. Declare the class as final so that it can’t be extended and its behaviour should not be overridden.
2. Make all fields private so that direct access is not allowed.
3. Initialize all the fields via a constructor performing **deep copy**.
4. Provide **only getter** methods for variables. Do not provide setters.
5. When getter method is called do not return actual reference of the object instead return copy of it.

Example: immutable. ImmutbleExample.java is there in String project

1. **Why string objects are immutable in java? Why String is made final in Java? IMP**
2. **String pool**: String is declared as final so that it can be immutable

Because java uses the concept of string literal and String pool concept.

Java has special implantation called String pool.

Say we are creating the string CHANDRA 10 times in our project but only one object will be created in the String pool.

1. **What is Race condition? IMP**

Race condition occurs when two or more threads access shared data and they try to change it at the same time without taking lock.

Here we do not know which thread access the data first because it all completely depends on thread scheduler.

1. **How will you avoid race condition?**

Race condition can be avoided by taking lock on shared resources.

Understand with example:

Problems often occur when one thread does a "**check-then-act**" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g:

If (x == 5) // The "Check"

{

y = x \* 2; // The "Act"

// If another thread changed x in between "if (x == 5)" and "y = x \* 2" above,

// y will not be equal to 10.

}

The point being, y could be 10, or it could be anything, depending on whether another thread changed x in between the check and act. You have no real way of knowing.

In order to prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time. This would mean something like this:

// obtain lock for x

if (x == 5)

{

y = x \* 2; // Now, nothing can change x until the lock is released.

// Therefore y = 10

}

// release lock for x

1. **What is volatile keyword in Java?**

When we use volatile keyword with a variable, all the threads read its value directly from the memory and not from cache.

This makes sure that the value read is the same as in the memory.

1. **Difference between ArrayList and LinkedList in Java**? **(I)**
2. **Implementation:**ArrayList is the resizable array implementation of list interface, while LinkedList is the Doubly-linked list implementation of the list interface.
3. **Reverse Iterator :**LinkedList can be iterated in reverse direction using descendingIterator() while there is no descendingIterator() in ArrayList , so we need to write our own code to iterate over the ArrayList in reverse direction.
4. **Memory Overhead:**Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node. While in ArrayList  each index only holds the actual object(data).
5. **Initial Capacity :** If the constructor  is not overloaded , then ArrayList creates an empty list of initial capacity 10 , while LinkedList  only constructs the empty list without any initial capacity.
6. **How Hash Map Works In Java? IMP**

HashMap works on the principle of **Hashing**.

**Hashing**: Hashing is technique where we can convert String or object to short fixed length Integer. This helps in faster indexing and look-up.

Before going further we should know three terms

1. **Hash Function**: The function which converts object/String to fixed integer value is called as hash function.
2. **Hash Value:** value returned by hash function is called as has value
3. **Bucket**: Bucket is used store key value pair. A bucket can have multiple key value pair. Bucket uses simple linked list to store key value pair.

When HashMap object is created by default array of Entry objects of size 16 will be created here. This Entry object array is called as bucket.

Entry is inner class of Map Entry object will have **key, value, hash, and Entry** (reference to another entry for linked list).

**Put (K, V);**

When put method is invoked with key and value.

1. **If key is null**: then it will always be inserted at index zero coz hash code of null is always zero.
2. **If key is not null:** First of all hash code of key will be computed and the returned hash code will be hashed again by the internal hash method to defend against poor hash code implementations.
3. Using the hash code generated and length of the bucket index at which key value pair to be stored will e calculated using bitwise &. (hash & (length - 1)).
   1. If there is no Entry object at the index calculated. The newly created Entry object for input key, value and hash will be pointed to the calculated index with Entry parameter as null.
   2. If there is an entry object already pointing to index calculated, newly created object will be inserted at the front to form linked list. New object will be inserted at the front in the linked list. (That is existing entry object will be pointed to newly created entry object’s entry reference and newly created entry object will be pointed to corresponding index of bucket. )
   3. If the input key is already present in the bucket then its value will be replaced.

**get (K);**

When get is called

1. **With null key :**  As it is already known fact that hash code of null is 0 . Key will be searched at index zero and if any key is matched with null corresponding value will be returned.
2. **If key is not null**: hash code of the key will be calculated and it will be further hashed and index will be calculated by using **hashcode & (length-1).** Once index is found all the Entry objects at that index will be looped (entry objects are stored as linked list form) by comparing hash and key once the match is found the corresponding value will be returned. If match is not found then null will be returned.

**remove(Key)**

As we know to find the desired Entry object which is to be removed in the HashMap we need hashValue , key and bucketindex . So remove(key) method calls  removeEntryForKey(key) method  internally , which calculate the final hashValue of the key object , and then use that hashValue in the indexFor(int,int) method to find the first entry object in the appropriate bucket.   
Since bucket(table) is a LinkedList effectively  , we start traversing from the first entry object which we got by using indexFor(int,int) method in the bucket. For each entry object in the bucket we compare whether  hashValue and the key is equal to the calculated hashValue in the first step and the key passed as a parameter in the remove(key) method.  
If desired Entry object is found , then we removed that single entry object from the LinkedList.   
Removing a single Entry object from the LinkedList is implemented just like removing a single object from the LinkedList.  
  
Entry object returned by the removeEntryForKey(key) method is then  stored in the local variable e of type Entry in the remove(key) method.  
  
If (e==null)  
     return null  
else  
    return value of removed Entry object.

1. **In which class hashCode method is there? IMP**

Object

1. **What is the contract between equals and hash code? IMP**  
   1. If two objects are equal, then they must have the same hash code.

2. If two objects have the same hash code, they may or may not be equal.

 If  object1.equals(object2) , then  object1.hashCode() == object2.hashCode() should always be true.

If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)

if (e.hash == hash && ((k = e.key) == key || key.equals(k)))

### ****Q1. What is a Lambda Expression and what is it used for****

In very simple terms, a lambda expression is a function that can be referenced and passed around as an object.

Lambda expressions introduce functional style processing in Java and facilitate the writing of compact and easy-to-read code.

Because of this, lambda expressions are a natural replacement for anonymous classes as method arguments. One of their main uses is to define inline implementations of functional interfaces.

### ****Q2. Explain the syntax and characteristics of a Lambda Expression****

A lambda expression consists of two parts: the parameter part and the expressions part separated by a forward arrow as below:

|  |  |
| --- | --- |
| 1 | params -> expressions |

Any lambda expression has the following characteristics:

* **Optional type declaration** – when declaring the parameters on the left-hand side of the lambda, we don’t need to declare their types as the compiler can infer them from their values. So int param -> … and param ->… are all valid
* **Optional parentheses** – when only a single parameter is declared, we don’t need to place it in parentheses. This means param -> … and (param) -> … are all valid. But when more than one parameter is declared, parentheses are required
* **Optional curly braces** – when the expressions part only has a single statement, there is no need for curly braces. This means that param – > statement and param – > {statement;} are all valid. But curly braces are required when there is more than one statement
* **Optional return statement** – when the expression returns a value and it is wrapped inside curly braces, then we don’t need a return statement. That means (a, b) – > {return a+b;} and (a, b) – > {a+b;} are both valid
  1. **Given an integers add+1 to it and find even number in it and sum it you can use java8 streams**