# Core Java:

|  |
| --- |
| **Q-1- Can we overload main () : Ans : - Yes** |
| Q2- How do we invoke the overloaded main method? And how does the java understand which main it has to call?  A: **Overloaded main method has to be called from inside the "public static void main(String args[])"** as this is the entry point when the class is launched by the JVM. Also overloaded main method can have any qualifier as a normal method have. |
| **3- What happens if i call a main method within another main method?** |
| **4- Is Null an Object in java?**  A: **null** is not an object, it is a primitive value. For example, you cannot add properties to it. Sometimes people wrongly assume that it is an object, because typeof null returns "object”. But that is actually a bug.  It is a reference type and its value does not refer to any object and so there is no representation of null in memory. No, null is **not a keyword**. The null is a literal similar to true and false in java, but it is a character string that is treated specially by the compiler |
| **5- What auto-boxing and un-boxing in java?**   1. [**Autoboxing**](https://www.geeksforgeeks.org/autoboxing-unboxing-java/) refers to the conversion of a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. 2. **Unboxing**on the other hand refers to converting an object of a wrapper type to its corresponding primitive value. For example conversion of Integer to int. |
| **6- Do you think System.gc () always prevent java from out of memory? OR**  **Does garbage collection guarantee that a program will not run out of memory?**  **No**, Garbage collection does not guarantee that a program will not run out of memory. The purpose of garbage collection (GC) is to identify and discard objects that are no longer needed by a Java program, so that their resources can be reclaimed and reused. An object is subjected to garbage collection when it becomes **unreachable** to the program in which it is used. |
| **7- What is the difference between pass by value and pass by reference?**  Pass by Value vs. Call by Reference. While calling a function, when you pass values by copying variables, it is known as “Call By Values.” While calling a function, in programming language instead of copying the values of variables, the address of the variables is used it is known as “Pass By References |
| **8- Let's say we have try-catch-finally block and we don't want to execute finally block, then how do we manage the code**  System. exit() can be used to avoid the execution of the finally block Finally Block. |
| **9- How Many types of class loader are there in java?**  The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. There are three different class loaders here: **application, extension, and bootstrap**. An application or system class loader loads our own files in the classpath. |
| **10- JVM Architecture** |
| **11- Difference Between JDK, JRE, and JVM** |
| **12- How the Garbage Collection works, how the java manages the memory internally.**  Memory management is the process of allocating new objects and removing unused objects to make space for new object allocations.  In Java, garbage collection is the process of managing memory, automatically. It finds the unused objects (that are no longer used by the program) and delete or removes them to free up the memory. The garbage collection mechanism uses several GC algorithms. The most popular algorithm that is used is Mark and Sweep.  [**https://docs.oracle.com/cd/E13150\_01/jrockit\_jvm/jrockit/geninfo/diagnos/garbage\_collect.html**](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html)  The following topics are covered:   * [The Heap and the Nursery](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1085825)  * [Object Allocation](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1085990)  * [Garbage Collection](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1086087)   **The Heap and the Nursery(Young Generation or Eden memory)**  Java objects reside in an area called the heap. The heap is created when the JVM starts up and may increase or decrease in size while the application runs. When the heap becomes full, garbage is collected. During the garbage collection objects that are no longer used are cleared, thus making space for new objects.  Note that the JVM uses more memory than just the heap. For example Java methods, thread stacks and native handles are allocated in memory separate from the heap, as well as JVM internal data structures.  The heap is sometimes divided into two areas (or generations) called the nursery (or young space) and the old space. The nursery is a part of the heap reserved for allocation of new objects. When the nursery becomes full, garbage is collected by running a special young collection, where all objects that have lived long enough in the nursery are promoted (moved) to the old space, thus freeing up the nursery for more object allocation. When the old space becomes full garbage is collected there, a process called an old collection.  The reasoning behind a nursery is that most objects are temporary and short lived. A young collection is designed to be swift at finding newly allocated objects that are still alive and moving them away from the nursery. Typically, a young collection frees a given amount of memory much faster than an old collection or a garbage collection of a single-generational heap (a heap without a nursery).  **Object Allocation**  During object allocation, JVM distinguishes between small and large objects. The limit for when an object is considered large, depends on the JVM version, the heap size, the garbage collection strategy and the platform used, but is usually somewhere between 2 and 128 kB.  Small objects are allocated in ***thread local areas* (*TLA*s).**  **Garbage Collection**  Garbage collection is the process of freeing space in the heap or the nursery for allocation of new objects. This section describes the garbage collection in the JRockit JVM.   * [The Mark and Sweep Model](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1085786)  * [Generational Garbage Collection](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1086786)  * [Dynamic and Static Garbage Collection Modes](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1086732)  * [Compaction](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html" \l "wp1086917)   **The Mark and Sweep Model**  **JVM** uses the mark and sweep garbage collection model or algo for performing garbage collections of the whole heap. A mark and sweep garbage collection consists of two phases, the mark phase and the sweep phase.  **During** the mark phase all objects that are reachable from Java threads, native handles and other root sources are marked as alive, as well as the objects that are reachable from these objects and so forth. This process identifies and marks all objects that are still used, and the rest can be considered garbage.  **During** the sweep phase the heap is traversed to find the gaps between the live objects. These gaps are recorded in a free list and are made available for new object allocation.  **Compaction**  To reduce fragmentation (free space), the JVM compacts a part of the heap at every garbage collection (old collection). Compaction moves objects closer together and further down in the heap, thus creating larger free areas near the top of the heap. The size and position of the compaction area as well as the compaction method is selected by advanced heuristics, depending on the garbage collection mode used. |
| **13- Solid Principal**  [**https://www.jrebel.com/blog/solid-principles-in-java#srp**](https://www.jrebel.com/blog/solid-principles-in-java#srp)  [**https://www.youtube.com/watch?v=BM\_lSZPMClo&t=14s**](https://www.youtube.com/watch?v=BM_lSZPMClo&t=14s)  SOLID principal is a five design principles in object-oriented programming to reduce code rot and improve the maintainability of application or we can say that it manage software design problem. The SOLID principles help the user develop loosely coupled code.  SOLID design is an acronym for the following five principles:   |  | | --- | | **1-**[**Single Responsibility Principle**](https://www.jrebel.com/blog/solid-principles-in-java#srp)  **Every class should have single responsibility**, in our code it should have only one job to do. It does **not** mean that your classes should only contain one method or property. There can be a lot of members as long as they relate to the single responsibility. If you have multiple responsibilities combined into one class, it might be difficult to change one part without breaking others. Mixing responsibilities also makes the class harder to understand and harder to test. **Example**  Let’s say we have class called BankService. This class contains multiple method   1. **public long** **deposit** (long amont, String accountNo) 2. **public long** **withDraw (**long amont, String accountNo**)** 3. **public void printPassbook()** 4. **public void sendOTP(** String medium**) { if (medimum.equals(“email) {}}** 5. **public void getLoanInterstInfo(**String loanType**)** : Here in this method we initially we are returning three type of loan interest information   if (loanType.equals(“homeLoan) {} , if (loanType.equals(“personalLoan) {}, if (loanType.equals(“car) {}  As we can see that this bank is supporting these thee loans now in future if introduce Gold loan or education loan then in this case we will have to modify this class and similarly currently OTP supports only for email, now if bank want to send OTP in mobile then again we will have to modify the existing code. So the above example does not fallow Single Responsibility Principal. **To make it as Single Responsibility we can move this into different services classes. So each service class contains single responsibilities to perform**.  Public void printPassbook() method can be moved in the Print Service , the loan related method can be moved into loan services and OTP method can be moved into notification service. Hence each service contains a single responsibility | | **2-**[**Open-Closed Principle**](https://www.jrebel.com/blog/solid-principles-in-java#ocp)**(OCP)**  **The Open-Closed Principle (OCP**) **states that classes should be open for extension but closed for modification.** “Open to extension” means that you should design your classes so that new functionality can be added as new requirements are generated. “Closed for modification” means that once you have developed a class you should never modify it, except to correct bugs.  We can achieve this by using interfaces or abstract classes, rather than using concrete classes. Functionality can be added by creating new classes that implement the interfaces.  **Example**:  As we can see in the above example in the sendOTP class we have two medium to send OTP. Now if in future we want to send OTP through wattsapp then in this case we should not modify the existing functionally but we can extend this class by including wattsapp medium. So in this case we will have to implement an interface having this functionality.  So to overcome the above problem either we will use abstract class or interface instead of using concrete method in the project.   1. Public void NotificationService(){   public void sendOTP(String medium);  public void sendTransactionReport()  }  Now create it implementation class according to need   1. Public class EmailNotificationService implements NotificationService {   // Here implement those two method defined in the NotificationService  }  Now create another class for Mobile OTP   1. Public class MobileNotificationService implements NotificationService{   // Here implement those two method defined in the NotificationService  }  Now create another class for Mobile OTP   1. Public class WhattsAppNotificationService implements NotificationService{   // Here implement those two method defined in the NotificationService  }  Now if in future if want to send OTP by any other medium then we just need to extend the interface instead of modification of any class. | | **3-**[**Liskov Substitution Principle**](https://www.jrebel.com/blog/solid-principles-in-java#lsp)**(LSP)**   1. This is an extension of Open Close Principal and applies on inheritance hierarchies 2. It says if Class A is a subtype of Class B, then object of type B can be replaced with object of type A. i.e. We should be able replace B with class A without interrupting the behaviour of the program i.e. **Derived Class must be completely substitutable for their base classes**     **Example: Let’s consider we have a abstract class called SocialMedia. This class contains several methods for all major activities like**   |  |  | | --- | --- | | public abstract class SocialMedia {  // @supported by Wattsapp, Facebook, Instagram  public abstract void chatWithFriend()**;**  // @Supported by Facebook, Instagram  public abstract void publishPost(Object post)**;**  // @Supported By Wattsapp, Facebook, Instagram  public abstract void sendPhotosAndVideos()**;**  // @Supported by Wattsapp, Facebook   public abstract void groupVideosCall()**;** } | **As here we can see that all the feature or method are supported by facebook so if we create a class facebook and extends this SocialMedia class then this facebook class can be substitute of the base class SocialMedia. But if want to implement another class like WhatsApp or Instagram then only few features are supporting. In this case Child class cannot be substituted with base class (SocialMedia) so this does not comply with LSP. So to resolve this issue we can create multiple interface and then we can extends or substitute those according to our need** | | | |  |  | | --- | --- | | public interface SocialCommonMedia { // @supported by WhatsApp, Facebook, instagram  public abstract void chatWithFriend()**;** // @Supported By WhatsApp, Facebook, Instagram public abstract void sendPhotosAndVideos()**;** } | public class FaceBook implements **SocialCommonMedia,PostMediaManager,VideoCallManager**{  @Override  public void chatWithFriend() {  }  @Override  public void sendPhotosAndVideos() {  }  @Override  public void publishPost(Object post) {   }  @Override  public void groupVideosCall() {   } }  **So here in case of FaceBook class we need to implement all the interface** | | public interface PostMediaManager {  // @Supported by Facebook, instagram public abstract void publishPost(Object post)**;** } | | public interface VideoCallManager {  // @Supported by WhatsApp, Facebook public abstract void groupVideosCall()**;** } | | public class Instagram implements **SocialCommonMedia,PostMediaManager**{  @Override  public void publishPost(Object post) {    }  @Override  public void chatWithFriend() {   }  @Override  public void sendPhotosAndVideos() {   } } | public class WhatsApp implements **SocialCommonMedia,VideoCallManager**{  @Override  public void chatWithFriend() {    }  @Override  public void sendPhotosAndVideos() {   }  @Override  public void groupVideosCall() {   } }  Here in case of Wattsapp we just need to implement two interface only according to our requirement |   **Hence the above solution satisfies LSP.**  **4-**[**Interface Segregation Principle**](https://www.jrebel.com/blog/solid-principles-in-java#isp)  As we know that in an interface all the method must be implemented form its derived (subclass) or client class. But this principal says that the larger interfaces must be split into smaller ones, so that the implementation classes use only the methods that are required. We should not force client to implement the method that it does not use. So **instead of creating one big interface we need to split it into many smaller and relevant interfaces so that we can use the only interface which is relevant to them. This principal was first used by Xerox company.**  **This principal is very similar to first principal**  **Example**: We can consider the above example where we have implemented the solution where we have segregated the abstract class into several interfaces according to our need, and hence we are not forcing our client class (like – Facebook, Whatsapp & Instagram to implement all the methods.  Another Example UPIPayment:   |  | | --- | | public interface UPIPayments {  public void payMoney()**;** public void getScratchCard()**;** // Does not supported by phonepe and paytm so let's create another interface having this feature  ///public void getCashBackAsCreditBalance(); } | | **public interface CashBackManager {  public void getCashBackAsCreditBalance(); }** | | **GPAY:**  **public class GPay implements UPIPayments,CashBackManager{  @Override  public void payMoney() {  }  @Override  public void getScratchCard() {  }  @Override  public void getCashBackAsCreditBalance() {  } }** | | **PAYTM:**  **public class Paytm implements UPIPayments{  @Override  public void payMoney() {  }  @Override  public void getScratchCard() {  } }** | | **PHONEPE:**  **public class Phonepe implements UPIPayments{  @Override  public void payMoney() {  }  @Override  public void getScratchCard() {   }  // Does not supported by phonepe // @Override // public void getCashBackAsCreditBalance() { // // } }**  **In this way we have segregated the interface and did not force the client to implement all the methods** | | | **5-**[**Dependency Inversion Principle**](https://www.jrebel.com/blog/solid-principles-in-java#dip)  This principal says that we must use abstraction (abstract classes & Interfaces) instead of concrete implementation. **High Level module should not depend on low level module but both should depend on abstraction.**  Use Case: When we go to any local store or any showroom to buy something and decide to by using card so when we give our card to cashier to make the payment then cashier does not bother about the card (Debit or Credit) that we have given, they just simply swipe it. It is an abstraction between cashier and us.  **Example**:   |  |  | | --- | --- | | public class DebitCard {  public void doTransaction(long amount){  System.*out*.println("Paid By Debit Card")**; }**  } | public class CreditCard {  public void doTransaction(long amount){ System.*out*.println("Paid By credit card")**;** } } | | public class Shopping { private DebitCard debitCard**;** public Shopping(DebitCard debitCard) {  this.debitCard = debitCard**;** } public void doPurchase(long amount){  debitCard.doTransaction(amount)**;** } public static void main(String[] args) {  DebitCard debitCard1 = new DebitCard()**;** Shopping shopping = new Shopping(debitCard1)**;** shopping.doPurchase(**5000**)**;** } | public class Shopping { private CreditCard creditCard**;** public Shopping(CreditCard creditCard) {  this. creditCard = creditCard**;** } public void doPurchase(long amount){  creditCard.doTransaction(amount)**;** } public static void main(String[] args) {  CreditCard debitCard1 = new CreditCard ()**;** Shopping shopping = new Shopping(creditCard)**;** shopping.doPurchase(**5000**)**;** } | |   As here we can see that the above Shopping class is completely tightly coupled we cannot use both the card in case of failure of anyone card and we had to do lot of changes. So it does not fallow the Dependency Inversion Principal.  **Solution**:   1. Let’s create on interface BankCard having the doTransaction(long amount) for debit and credit card. 2. Implement this interface now in DebitCard and CreditCard classes 3. Now in the Shopping main class instead of injecting DebitCard or CreditCard, we will simply inject BankCard because this contains the implementation of both DebitCard and CreditCard   So the above class now will look like below   |  |  | | --- | --- | | public interface BankCard {  public void doTransaction(long amount)**;** } | | | public class DebitCard implements **BankCard**{  public void doTransaction(long amount){  System.*out*.println("Paying using Debit Card")**;** } } | public class CreditCard implements **BankCard**{  public void doTransaction(long amount){ System.*out*.println("Paying using credit card")**;** } } |   **public class ShoppingSolution {   private BankCard bankCard;   public ShoppingSolution(BankCard bankCard) {  this.bankCard = bankCard;  }  public void purchase(long amount){  bankCard.doTransaction(amount);  }   public static void main(String[] args) {   //BankCard bankCard1= new DebitCard();  BankCard bankCard1= new CreditCard();  // only above line needs to change at run time, called runtime polymorphysm**  **ShoppingSolution solution = new ShoppingSolution(bankCard1);  solution.purchase(5000);   }**  Now the above code is not tightly coupled and now we can pay using Debit and Credit card just by making changes at one place at runtime. |
| **14- Internal working of HashMap, How the hashCode and equal method help in HashMap**  How the HashMap work internally?   |  |  | | --- | --- | | **package** com.employee.salary;  **import** java.util.HashMap;  **import** java.util.Map;  **public** **class** HashSetInternalWorking {  **public** **static** **void** main(String[] args) {  Map<String, Integer> hsMarks = **new** HashMap<String, Integer>();  hsMarks.put("Naveen", 100);  hsMarks.put("Tom", 200);  hsMarks.put("Lisa", 300);  hsMarks.put("Peter", 400);  hsMarks.put("Ruby", 600);  hsMarks.put(**null**, 100);  hsMarks.put(**null**, 100);  hsMarks.put("Naveen", 900);  }  } |  |  1. As here in the above program we have created one HashMap<String, Integer> and using this HashMap object we are inserting name as string and marks in integer. 2. Since in HashMap null key and duplicate is allowed that is why we have created two null entries and two duplicate entries. 3. HashMap is data structure which stores the data in the form of key-value pair. HashMap work on basis if hashing. Java using one method called hasCode() which belongs to Object class. Let’s understand how the HashMap works. 4. The moment we create HashMap it creates 16 nodes (0-15) or bucket by default. The default value of the HashMap is 16. 5. Each node or bucket will maintain [hashCode, Key, Value, PointerOfNextNode]. 6. Now on the basis of hashing concept java will start storing these values in the bucket. So very first time when we will make an entry like this [hsMarks.put ("Naveen", 100);] then first of all a hashCode will be calculated using the HashMap key [int hash= hashCode (Naveen)] and it will provide an integer number. Let’s say it gives an integer hashCode value (210687). 7. Now using this integer hashCode we will be calculating index using [int index= hash & (n-1)]. Here n is size of the HashMap ( like here initially the default size of HashMap is 16)   **Note: Q:** Why we can’t use this hashCode as an index because here these buckets is in the form of index array.  **Ans**: The hashCode that we have got is so big in number and practically it is not possible to use as a because if we using this hashCode as an index then it will consume lot of memory and very soon we will be out of memory. That is why using this hashCode we calculate an index. Let’s say for the first entry [Naveen] we have got index =4   1. Now the first entry will be stored at index 4 in the form of **node[hashCode, Key, Value, PointerOfNextNode]**. If it does not have any next node at index 4 then this means it is not pointing to any next node and in this case PointerOfNextNode will null. So the exact node will be look like [210687, Naveen, 100, null] 2. Similarly let’s assume for the next entry [hsMarks.put("Tom", 200);] the hashCode is 210780 and the index is 9 then this entry will be stored at index 9 in the form of **node[**210780**, Tom, 200, null]**. 3. Now let’s assume one case where two entry has got same hashCode, then in this case the index number also will be same and hence that entry then will be stored on the same index and in this case at one index we will have two entry or node and then in this case **PointerOfNextNode** will not be null and instead of null this node will point the address of next node. Let’s assume for entry [hsMarks.put("Lisa", 300);] we have got exactly same hashCode what we have got in case of first entry means hashCode= 210687 and index is 4 then in this case this record will be stored at index 4 and in this case the both the node will look like as [210687, Naveen, 100, **PointerOfNextNode**] 🡪 [210687, Lisa, 300, **null**]. So in this case it behaves like linked list. 4. Similarly for the remaining entries the hashCode will be calculated using key and further using hashCode index will be calculated and then entries will be stored at the same index. Sometime it could be possible that we will get different hashCode but will get same index then in that case entry will be stored at the same index next to the already available entry in the form of linked –list.   **Note: When several entries are stored at the same index then this condition is called collision**    **Let’s talk about null key and duplicate key:**   |  |  | | --- | --- | | hsMarks.put(**null**, 100);  hsMarks.put(**null**, 100); | In case of Null key the calculated hashCode always will be zero (0) and then index also will be zero and hence in case of null the entry always will be stored at zeroth index in HashMap and the entry will like **node[hashCode, Key, Value, PointerOfNextNode]**🡨🡪 **node[**0**, null, 200, null]**. | | hsMarks.put("Naveen", 900); | In case of duplicate key again the hashCode and index will be same and then in this case the old value will be replaced with new value for the duplicate key |   **Let’s talk about fetching value from HashMap**. Since we know that in case of HashMap the value will be fetched by Key. So it works internally?  Let’s say we are trying to fetch hsMarks.get("Naveen");  In this case also first of all the hashCode will be calculated by using key and then further using that hashCode, index will be calculated, let’s say we have got hashCode= 210678 and index = 4 . Now it will go to index four and here it will found three nodes. So first it will go first node and check whether we have the same hashCode using **equals () method** of Object class. If hashCode got matched then it will search for the same key and if key is matched with the same hashCode then the corresponding value will be fetched. Like here in case of Naveen key 100 as value will be fetched. Similarly in case of Lisa we will get same hashCode and index, so in this case now it will go to index 4 and first it will check the first node and here it will found that hashCode is same but key is not matched so now it will move to the next available node and again check and compare using equal () method and now here it will found the Key Lisa at the same hashCode and then in this case value 300 against Lisa Key will be fetched. So this is how get operation works.      It means if the nodes having same hashCode keep adding at the same index and let’s say it reaches to its threshold then this case the linear linked-list will be automatically converted into balanced tree to improve the performance.  The performance of Linked-list is O(n) and for balanced tree O(log n)  Note: As we know that the initial size or capacity of HashMap is 16 (0-15). And its factor value is .75. It means once the 75% (threshold value) HashMap gets occupied then the value of the HashMap will be double in the size. The threshold value will be calculated as 16x.75= 12 means once the entry reaches to 12 index then HashMap size will become double. |
| 15 Why to Override equals(Object) and hashCode() method ?Ans: <https://www.geeksforgeeks.org/override-equalsobject-hashcode-method/> **15- We have overridden hashCode and equal method for employee object and in hashCode we are always return 1. Now i am inserting 5 element in the HashMap object then what will be the size** |
| **16- What impact will be there if we do not override hashCode () but override equals () method, and in opposite scenario if we override hashCode and does not override equals ().**  **HashCode():** As we know in the java Object is the parent class of all the object present in the java and every object extends this Object class directly or indirectly.  If we go inside the Object class then we will get one method of hashCode () declared like this [**public native int hashCode ();**].  So here in the declaration we can see that hashCode method is written is declared as native i.e. it is written or implemented in native language either in C or C++ and while overload this hashCode() method in our own user defined java class then the native method code of hashCode is gets overloaded.  And we it is not forcing to implement this in our own user defined java class. Note: This native method is also called **foreign** method.  This hashCode method return an integer number in other word Object Class hashCode() method returns an unique number for each object.  So we can say that every object can be identified by its hashCode number just like a human being in India can be identified by his aadhar number.  Q: How we get a unique number all the time for every object? Is this the memory address of the object?  A: As we know when we create an object then it is created in heap and inside the heap where the object is stored, contains the address and this address and hashCode both are different number for the object. For example the aadhar number and the address of the person in the India both are different things.  But the one thing which is true is that this hashCode is getting generated by using the memory address of the object.  We cannot not get or see the memory address of the object because this is taking care by JVM itself.  So in short: Each object has a unique memory address which helps the Object class hashCode() method to generate a unique integer hashCode number for each object.  Q: Can we overriede this hashCode() method and can provide our own implementation.   1. Yes   Note: \* As much as is reasonably practical, the hashCode method defined by  \* Class {@code Object} does return distinct integers for distinct  \* Objects. It is mandatory that every object contains unique distinct integers hashCode number,   |  |  | | --- | --- | | **package** com.employee;  **public** **class** Employee {  **private** **int** aadharNumber;  **private** String name;  **public** Employee(**int** aadharNumber, String name) {  **super**();  **this**.aadharNumber = aadharNumber;  **this**.name = name;  }  } | **package** com.employee;  **public** **class** TestEmployeeHashCodegeneration {  **public** **static** **void** main(String[] args) {    Employee emp1 = **new** Employee(1001, "Arun");  Employee emp2 = **new** Employee(1002, "Tarun");  **int** emp1HashCode = emp1.hashCode();  **int** emp2HashCode = emp2.hashCode();  System.***out***.println("emp1HashCode="+ emp1HashCode);  System.***out***.println("emp2HashCode="+ emp2HashCode);  String str1 = "FB";  String str2 = "EA";  System.***out***.println("str1HashCode="+ str1.hashCode());  System.***out***.println("str2HashCode="+ str2.hashCode());  }  } | | **Output**:  emp1HashCode=366712642  emp2HashCode=1829164700  str1HashCode=2236  str2HashCode=2236 | **As here in output we can see that emp1 & emp2 has distinct unique hashCode number but str1 & st2 has same unique hashCode number.**  **But a good hashCode() should return a distinct integer for each object as much as possible.** |   Creating our own custom hashCode |
| **8- How does the hash map put and get method work.**  **See the answer above --** |
| **9- What is SSL?**  SSL now technically known as TLS (Transport Layer Security), it encrypts the communications or connection between 2 endpoints for secure data exchange.      **Points to be noted:**   * As per the diagram we can see that the Kafka server (technically called Kafka broker) contains Server certificate (Digital cert) signed by CA. This signed certificate is stored in Key-Store * When the Kafka-client send any request to Kafka Server then first Kafka Server sends one signed SSL certificate to the Kafka-client   **9- How does the certificate work in java when you expose services on SSL?**  [**https://www.youtube.com/watch?v=33VYnE7Bzpk**](https://www.youtube.com/watch?v=33VYnE7Bzpk)  A: When we are doing any online shopping or banking transaction, we want to make sure it is HTTPs, and a green padlock icon is in the address bar.  What does the HTTPs means? What is significant about that small green padlock?  To answer these two questions, we need to understand SSL certificate and how it works.   * HTTPS is the secure version of HTTP protocol used between a browser and a web server. The “S” stands for secure. Technically it refers to HTTP over a Secure Socket Layer (SSL) * HTTPS means all communications between your browser and web server are encrypted * Behind HTTPs, SSL certificate plays an important role in building trust between a browser and a web server  |  |  | | --- | --- | |  |  |  * By definition, a SSL certificate is a web server’s digital certificate issued by a third party , and verifies the identity of the web server and its public key   Let’s take an example to understand: I want to connect with yahoo web server and I want all communication with yahoo web server are encrypted. I type in <https://www.yahoo.com> , now what happen when I hit enter?   |  | | --- | | Step1- My browser requests secure pages(HTTPs) from yahoo web server | | Step2- The Yahoo Server sends its public key with its SSL certificate, which is digitally signed by a third party or Certificate Authority (CA) | | Step3- Once my browser gets the certificate, it will check the issuer’s digital signature to make sure that the certificate is valid.  As we know a Digital signature is created by a CA’s private key and my browser like (Chrome or Firfox) is priviously installed with many major CA’s public keys.In other words these CA’s public is already installed in browser . Thus digital signature can be verified. Once the certificate’s signature is verified, the digital certificate can be trusted and after that a green padlock icon appears in the address bar. The green padloc simply indicates that the web server’s public key really belongs to the web server, not someone else. Now verification is done! And now its time to exchange a secrete. | | **step 4,**    To exchange secrete, now browser creates one symmetric key, a shared secret. It keeps one and gives a copy to the web browser. However my browser does not want to send the shared secret in plain text. Therefore it uses the web server’s public key to encrypt the secret, and sends it to the web server. | | **Step-5** When the web server gets the encrypted symmetric key, it uses its private key to decrypt it. Now the web server gets the browser’s shared key**. From now on, all traffic between the client and web server will be encrypted and decrypted with the same key (Symmetric Key)**  **Note: In the above example, we actually demonstrate how asymmetric key algorithm and symmetric key algorithm works together.**   * **Asymmetric Key (Public & Private) is used to verify the identity of the owner and its public key so that trust is built. Once the connection is established symmetric key algorithm (Shared Key) is used to encrypt and decrypt all the traffic between them.** * **Keep in mind, https: and green padlock only indicate the communications between client and server are encrypted. It does not mean the website itself is safe and secure. Anyone including hacker can get a SSL certificate for their sites.**   [**https://www.youtube.com/watch?v=UbMlPIgzTxc&list=PLSNNzog5eyduzyJ8\_6Je-tYOgMHvo344c**](https://www.youtube.com/watch?v=UbMlPIgzTxc&list=PLSNNzog5eyduzyJ8_6Je-tYOgMHvo344c) | |
| **8- What is the Singleton DP, write a program with all scenarios, how do you test that singleton class that you have created is singleton class only.** |
| **final Keyword:**   |  |  | | --- | --- | |  | Final keyword can be used with variable, method, class. |   **Final variable:**   1. As we know that the instance variable varies from object to object. But if use final with instance variable then it is treated as constant. 2. For instance variable we are not required to perform initialization explicitly, JVM will always provide default values if we don’t initialize instance variable at the time of declaring. 3. If the instance variable declared as final then compulsory we have to perform initialization explicitly whether we are using or not and JVM won’t provide default values.  |  |  |  | | --- | --- | --- | | **Class test{**  **Int x;**  **Static Int y;**  **P s v main (String[] args) {**  **Test t = new Test();**  **Sopln(t.x); Sopln(t.y);**  **}**  **Output: 0 for x and y both (default value of the integer)** | **Final variable is not initialized at declaration time**  **Class test{**  **final Int x;**  **P s v main (String[] args) {**  **Test t = new Test();**  **Sopln(t.x);**  **}**  **CE: Variable x must be initialized** | **At the time of variable declaration.**  **Class test{**  **final Int x=10;**  **P s v main (String[] args) {**  **Test t = new Test();**  **Sopln(t.x);**  **}**  **Output: 10** | | **Inside the instance block.**  **Class test{**  **final Int x;**  **{**  **X=10**  **}**  **Output: 10** | **Inside constructor:**  **Class test{**  **final Int x;**  **test() {**  **X=10**  **}**  **Output:10** | **Do not initialize inside method.**  **Class test{**  **final Int x;**  **final static Int y;**  **public void(){**  **x=10; y=20;**  **}**  **CE: cannot assign final Variable x & y inside method, because method will be executed after constructor.** |   Rule1-: For finale variable compulsory perform initialization before constructor completion.  Rule2-: Finale variable can be initialized inside the instance block  Rule3-: Final variable can be initialized inside the constructor  These are the only places to perform initialization for final variable. If we are trying to perform initialization anywhere else then we will get compile time error.  Note: if the value of variable is not going to change object to object then never declare the variable as an instance variable because it will waste unnecessary memory and decrease the performance. For example the College name for all the student is not going to change. Such type of variable should be declared as static at the class level. And hence only once copy will be created in the memory and shared by the entire object of the class. For static variable the default value is going to provide by JVM, we are not required to perform static variable initialization explicitly.  Order of modifier is not important:  Static final int x=0 or final static x =0 both are same.  Q: But if static variable is final then what will happen?  So compulsory we must perform initialization explicitly, otherwise compile time error we will get and JVM would not provide any default values. And three rules final static variable.  Rule1-: For finale variable compulsory perform initialization at the time declaration before class load completion i.e. before main () loading.  Rule2-: Finale variable can be initialized inside the static block.  Apart from these two places if we perform initialization anywhere else then we will get compile time exception.   |  |  |  | | --- | --- | --- | | **Do not initialize inside method.**  Class test{  final Int x;  final static Int y;  public void(){  x=10; y=20;  }  CE: cannot assign final Variable x & y inside method, because method will be executed after constructor | **Inside the static block.**  Class test{  final Int x;  static{  X=10;  }  Output: 10 | **At the class level before main () method gets loaded.**  Class test{  Final static Int x=10;  P s v main (String[] args) {  Test t = new Test();  Sopln(t.x);  }  Output: 10 | |

9- WAP to make an Employee class immutable.

10- Let's say we have one class with two variables (name, age) and one variable of class type let's say Engine class type then how will you make this class immutable?

11- What is the difference between fail fast and fail safe, what kind of collection is fail fast and fail safe?

10- WAP to count the number of given character in the string (My Country is India) using java-8

11- I have one array list having element 1 to 15, then using java find the number of multiple of 5.