**12 Interview Question and Answer**

# 1. What is Immutable class?

Immutable class means a class whose properties we can’t modify if we try to modify also it will always create new instance means simply unchangeable.

# 2. How to create our own immutable class and reason?

There is 2 ways we can create customized immutable class

**Approach: 1**

**package** com.string.prog;

**final** **class** MyImmutable {

**private** **int** id;

**private** String name;

**private** String dept;

**private** **double** salary;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDept() {

**return** dept;

}

**public** **void** setDept(String dept) {

**this**.dept = dept;

}

**public** **double** getSalary() {

**return** salary;

}

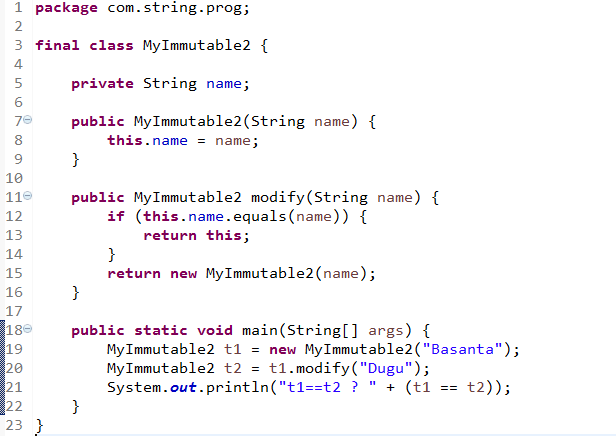
**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

}

**Approach: 2**

****

**NOTE: Here in second approach I didn’t instantiate t2 but Boolean will return false cause due to customized immutable.(**Reason in project we have some class if we modified it then it will effect on multiple class which is dependent on that class so we need this situation we have to make this class as immutable**)**

# 3. Difference between hashcode() and equals() method?

Hashcode method meant for grouping the object whose return type is int and equals method is meant for content comparison whose return type is Boolean both are present in java.lang package.

But normally both methods following contract that’s why we have to override both normally when we add object in collection then it’s recommended to override both to avoid duplicate.

# 4. Difference between HashMap and ConcurrentHashMap?

|  |  |
| --- | --- |
| ***HashMap*** | ***ConcurrentHashMap*** |
| 1.HashMap is non-synchronized and non-thread safe | 1.ConcurrentHashMap is synchronized and thread safe |
| 2.It introduced in java 1.2 in java.util package | 2.It introduced in 1.5 in Concurrent package |
| 3.It apply locks on complete object so there may be chance for getting concurrentModificationException when multiple thread try to access same Collection object | 3.It apply locks on single entry so is no concurrentModificationException when multiple thread try to access same Collection object |
| 4.it internally use HashTable | 4.It internally used frenced grained mechanisim. |

# 5. List of employee we have to sort it by name and age?

**package** com.collection.prog;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.Date;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.TreeSet;

**public** **class** Employee **implements** Comparable<Employee> {

**public** String name;

**public** **int** age;

**public** **double** sal;

**public** Date dob;

**public** Employee(String name, **int** age, **double** sal, Date dob) {

**super**();

**this**.name = name;

**this**.age = age;

**this**.sal = sal;

**this**.dob = dob;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + age;

result = prime \* result + ((dob == **null**) ? 0 : dob.hashCode());

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**long** temp;

temp = Double.*doubleToLongBits*(sal);

result = prime \* result + (**int**) (temp ^ (temp >>> 32));

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Employee other = (Employee) obj;

**if** (age != other.age)

**return** **false**;

**if** (dob == **null**) {

**if** (other.dob != **null**)

**return** **false**;

} **else** **if** (!dob.equals(other.dob))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**if** (Double.*doubleToLongBits*(sal) != Double.*doubleToLongBits*(other.sal))

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Employee e) {

**return** **this**.age - e.age;

}

@Override

**public** String toString() {

**return** "Employee [name=" + name + ", age=" + age + ", sal=" + sal + ", dob=" + dob + "]";

}

**public** **static** **void** main(String[] args) {

List<Employee> emps = **new** ArrayList<>();

Set<Employee> s = **new** TreeSet<>(**new** NameSortComparator());

s.addAll(emps);

s.add(**new** Employee("Basanta", 24, 12345, **new** Date()));

s.add(**new** Employee("Bikash", 28, 36483, **new** Date()));

s.add(**new** Employee("Manoj", 21, 68056, **new** Date()));

s.add(**new** Employee("Rakesh", 18, 345, **new** Date()));

System.***out***.println(s);

}

}

**package** com.collection.prog;

**import** java.util.Comparator;

**public** **class** NameSortComparator **implements** Comparator<Employee> {

@Override

**public** **int** compare(Employee e1, Employee e2) {

**return** e1.name.compareTo(e2.name);

}

}

# 6. Difference between HashSet and LinkedHashSet?

The only difference between HashSet and LinkedHashSet is that LinkedHashSet maintains the insertion order. When we iterate through a HashSet, the order is unpredictable while it is predictable in case of LinkedHashSet.

The reason why LinkedHashSet maintains insertion order is because the underlying data structure is a doubly-linked list.

# 7. Is there any Collection where we can store object in key and value pair and key has to be sorted?

As I know there is no Collection implementation where we can store object in key value pair but we can manually do it by following any Map implementation class we can add object then we need to add this complete map object into List as below.

Map<Object,Object> map=**new** HashMap<>();

List<Map.Entry<Object,Object>> list=**new** ArrayList<>();

# 8. Difference between wait and sleep?

**Sleep ():**

It is a static method on Thread class. It makes the current thread into the "Not Runnable" state for specified amount of time. During this time, the thread keeps the lock it has acquired.

**Wait ():**

It is a method on Object class. It makes the current thread into the "Not Runnable" state. Wait is called on a object, not a thread. Before calling wait () method, the object should be synchronized, means the object should be inside synchronized block. The call to wait () releases the acquired lock

# 9. What is the collection used in your project?

As per my project business requirement we used ArrayList and HashMap and properties collection

ArrayList-in DAO layer

Map- in Service layer

Properties-separate file to avoid hard coded value.

# 10. Can I add Employee Object in HashMap?

***Employee.class***

**package** com.collection.prog;

**public** **class** Employee2 {

**public** **int** id;

**public** String name;

**public** String dept;

**public** Employee2(**int** id, String name, String dept) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.dept = dept;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + ((dept == **null**) ? 0 : dept.hashCode());

result = prime \* result + id;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Employee2 other = (Employee2) obj;

**if** (dept == **null**) {

**if** (other.dept != **null**)

**return** **false**;

} **else** **if** (!dept.equals(other.dept))

**return** **false**;

**if** (id != other.id)

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "Employee2 [id=" + id + ", name=" + name + ", dept=" + dept + "]";

}

}

**Test class:**

****

# 11. How Collision will occur in HashMap?

Normally when we add object in HashMap it internally calls hashcode to find the bucket or index in internal linked list to store object so based on index object will placed .if in same index more than one object will placed with having same hashcode then this condition we can say hashing collision so to avoid it hash map internally calls equals method for content comparison if it returns true then value will be replaced with existing value else it will added object in same object