♦ Java immutable objects interview questions & answers

Posted on August 16, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Best Practice: “Classes should be immutable unless there’s a very good reason to make them mutable….If a class cannot be made immutable, limit its mutability as much as possible.” — by Joshua Bloch

Q1. What is an immutable object?

Q2. Immutable objects are objects whose state (the object’s data) cannot change after construction. Examples of immutable objects from the JDK include String and wrapper classes like Integer, Double, Character, etc.

9 tips to earn more | What can u do to go places? | **945+** members. LinkedIn Group. **Reviews**

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- Ice Breaker Interview
  - 01: ♦ 15 Ice breake
  - 02: ♥♦ 8 real life ex
  - 03: ♦10+ Know you
  - 04: Can you think c
  - 05: ♥ What job inte
  - 06: ► Tell me abou
  - 07: ♥ 20+ Pre inter
- Core Java Interview C
  - Java Overview (4)
    - 01: ♦ ♥ 17 Java c

Q2.How do you create an immutable type?

A2.

1. Make the class **final** so that it cannot be extended or use static factories and keep the constructors private.

```
1 public final class MyImmutable { … }
```

2. Make the fields private and final.

```
1 private final int[ ] myArray;
```

3. Don't provide any methods that can change the state of the immutable object in any way from outside the object – not just *setXXX* methods, but any methods which can change the state.

4. The "**this**" reference is not allowed to escape during construction from the immutable class, and the immutable class should have exclusive access to fields that contain references to other mutable objects like arrays, collections and mutable classes like Date by:

a) Declaring the mutable references as private.
b) Not returning or exposing the mutable references to the caller. This can be done by defensively copying the objects by deeply cloning them.

Example: satisfying the above conditions

```
1   public class User {
2      private final String firstName; //final, and
3      private final String lastName;  //final, and
4
5      // constructor is private
6      private User(String firstName, String lastNa
7          this.firstName = firstName;
8          this.lastName = lastName;
9      }
10
11     //factory method
12     public static User getInstance(String firstN
13         return new User(firstName, lastName);
14     }
```

```
15
16      //only getters, no setters
17
18      public String getFirstName() {
19          return firstName;
20      }
21
22      public String getLastName() {
23          return lastName;
24      }
25
26  }
```

Q3. Is the following class immutable?

```
1  import java.util.Arrays;
2
3  public final class MyImmutable {
4
5      private final Integer[ ] myArray;
6
7      public MyImmutable(Integer[ ] anArray) {
8            this.myArray = anArray;
9      }
10
11     public Integer[ ] getMyArray( ) {
12         return myArray;
13     }
14
15     //....equals(), hashCode(), etc
16
17     @Override
18     public String toString() {
19         return Arrays.toString(myArray);
20     }
21 }
```

A3. No. The above class is **not immutable** as it fails **#4** condition where the “myArray” reference can escape, and mutated from outside as demonstrated below.

```
1  public class MyImmutableTest {
2
3      public static void main(String[] args) {
4          Integer[] array1 = {1,2,3};
5          MyImmutable mi = new MyImmutable(array1)
6          System.out.println("before modifying: "
7
8          mi.getMyArray()[2] = 4; //change 3 to 4
9          System.out.println("after modifying: " +
10     }
11 }
```

Output:

```
1 before modifying: [1, 2, 3]
2 after modifying: [1, 2, 4]
```

FIX: "myArray" reference is deeply copied, and can't escape

```
1  import java.util.Arrays;
2
3  public final class MyImmutable {
4
5      private final Integer[] myArray;
6
7      public MyImmutable(Integer[] anArray) {
8          this.myArray = anArray.clone(); //cloned
9      }
10
11     public Integer[] getMyArray() {
12         return myArray.clone(); // cloned array
13     }
14
15     // ....equals(), hashCode(), etc
16
17     @Override
18     public String toString() {
19         return Arrays.toString(myArray);
20     }
21 }
```

If you run the "MyImmutableTest" again,

```
1 before modifying: [1, 2, 3]
2 after modifying: [1, 2, 3]
3
```

Q4. What are the advantages of immutable objects?
A4.

1) Immutable classes can greatly simplify programming by freely allowing you to cache and share the references to the immutable objects without having to defensively copy them or without having to worry about their values becoming stale or corrupted.

2) Immutable classes are inherently thread-safe and you do not have to synchronize access to them to be used in a multi-threaded environment. So there are no chances for negative

performance consequences as multiple threads can share the same instance.

3) Eliminates the possibility of data becoming inaccessible when used as keys in HashMaps or as elements in Sets. These types of errors are hard to debug and fix.

4) Eliminates the need for class invariant check once constructed.

5) Allow hashCode( ) method to use lazy initialization, by caching its return value.

6) Cloning is not required.

7) Simpler to construct, use, and test due to its deterministic state.

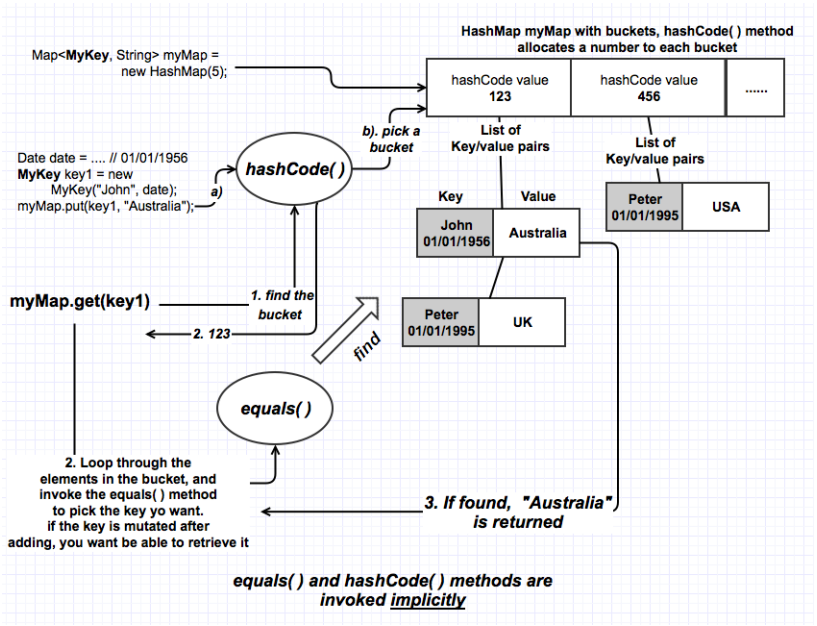
Q5. Why is it a best practice to implement the user defined key class as an immutable object?
A5. Immutable objects generally make the best map keys as the keys cannot be modified once they have been added to the Map. In general *String*, *Integer*, or *Long* are used as keys, which are immutable objects. If you define your own key class, make sure that they are immutable. Otherwise, if the keys are accidentally modified after adding to a Map, you will never be able to retrieve the stored value as the key values have been changed. This is a common pitfall many Java developers, especially beginners fall for.

Example:

Immutable key class

```
1
2  import java.util.Date;
3
4  public final class MyKey {
5
6      private final String name;
7      private final Date myDate;
8
9      public MyDiary(Date aDate) {
```

```
10          this.myDate = new Date(aDate.getTime( ))
11      }
12
13      public Date getDate( ) {
14          return new Date(myDate.getTime( )); //de
15      }
16
17      public String getName( ) {
18          return name; //String is immutable
19      }
20
21      //...equals( ), hashCode(), etc
22  }
23
```



hashCode( ) Vs. equals( )

As shown, when Maps are used in Java, the **equals( )** *and* ***hashCode( )*** methods are implicitly invoked. If these methods are incorrectly implemented or the keys are modified once added to the map, then unpredictable behavior will be experienced, and these behaviors are harder to debug. The hashCode( ) and equals( ) methods are implicitly invoked to determine where the key is stored, and to retrieve the value for a particular key respectively. More than one key/value pairs can be stored in the same bucket.

The *hashCode( )* method does not give a unique value each time. Its duty is to spread out the numbers so that your key value pairs get spread out in multiple buckets. So, always remember this.

The *hashCode( )* method is used to store the key/value in a bucket, and both the hashCode() and equals() methods are called to retrieve the stored key/value. If they are implemented inconsistently or the key is mutated, then the stored object cannot be retrieved as the returned values of these methods will vary in between different invocations. To be more specific, the hashCode( ) method is called to determine the key index (aka the bucket) of the array, and the equals( ) method is called to retrieve the exact key from the list of keys belonging to that particular key index (or bucket) as the same bucket will be holding multiple keys linked to multiple values. Remember the contract between these two methods?

"If 2 objects are equal, they must return the same hashCode( ) value, but the reverse is not true. Which means, if 2 objects return the same hashCode( ) value does not mean that those 2 objects are equal( )".

Q6. How would you defensively copy a Date field in your immutable class?

A6.

```
1  public final class MyDiary {
2
3      private final Date myDate;
4
5      public MyDiary(Date aDate) {
6          this.myDate = new Date(aDate.getTime( ))
7      }
8
9      public Date getDate( ) {
10         return new Date(myDate.getTime( )); //de
11     }
12 }
13
14
```

Q7. How will you prevent the caller from adding or removing elements from a collection of pets?

A7.

```
1
2  import java.util.ArrayList;
3  import java.util.Collections;
4  import java.util.List;
```

```
5
6  public class PetCage {
7      private final List<Pet> pets;
8
9      public PetCage(List<Pet> pets) {
10           this.pets = Collections.unmodifiableLi
11     }
12
13     public List<Pet> getPets( ) {
14           return pets;
15     }
16 }
17
```

It ensures that you cannot add or remove pets. However, there is no guarantee that the pets are also immutable. To make this instance fully immutable, the Pet instance itself must be immutable or use the decorator pattern as a wrapper around each of the pets to make them also immutable. For example, The *Integer* **wrapper class provides immutability to mutable primitive int value** . You could also defensively deep copy the list of pets in the constructor and getPets( ) method.

Q8. How about data that needs to be mutable, but less frequently? Is there any way to obtain the benefits of immutability with the added benefit of thread-safety for data that changes less frequently?
A8. The **Copy-On-Write** collections like *CopyOnWriteArrayList* and *CopyOnWriteArraySet* classes introduced from JSE 5.0 util.concurrent package are good examples of how to harness the power of immutability whist permitting occasional modifications for infrequently changing data. CopyOnWriteArrayList behaves much like the ArrayList class, except that when the list is modified, instead of mutating the underlying array, a new array is created and the old array is discarded. CopyOnWriteArrayList is designed for cases where:

— reads hugely outnumber writes.
— the array is small (or writes are very infrequent).
— the caller genuinely needs the functionality of a list rather than an array.

When you obtain an iterator, which holds a reference to the underlying array, the array referenced by the iterator is

effectively immutable and therefore can be traversed without synchronization or risk of concurrent modification. This eliminates the need to either clone the list before traversal or synchronize on the list during traversal. If reads are much more frequent than insertions or removals, which is the case very often, the Copy-on-Write collections and *ConcurrentHashMaps*offer better performance and development convenience. The development convenience is provided not needing to deal with synchronization, deep cloning, or "ConcurrentModificationException". The ConcurrentModificationException is generally thrown by an ArrayList, HashSet, or a HashMap implementation when you try to remove an object from a collection while iterating over it.

Q8. Can builder design pattern be used to create immutable objects?
A8. Yes.

Q9. Can you give a builder design pattern example to create immutable objects?
A9. **Example**: Builder pattern and immutability in Java.

Popular Posts

♦ 11 Spring boot interview questions & answers

857 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

825 views

18 Java scenarios based interview Questions and Answers

447 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

401 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

302 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

292 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ Java serialization, cloning, and casting interview Q&A

♦ 5 Java Object class methods interview questions & answers ›

Posted in member-paid, Objects

Tags: Core Java FAQs, Java/JEE FAQs, Novice FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

- Unit Testing Q&A (2
  - BDD Testing (4)
    - Java BDD (Be
    - jBehave and E
    - jBehave and j
    - jBehave with t
  - Data Access Uni
    - ♥ Unit Testing
    - Part #3: JPA H
    - Unit Test Hibe
    - Unit Test Hibe
  - JUnit Mockito Sp
    - JUnit Mockito
      - Spring Con
      - Unit Testing
    - Part 1: Unit te
    - Part 2: Mockit
    - Part 3: Mockit
    - Part 4: Mockit
    - Part 5: Mockit
  - Testing Spring T
    - Integration Un
    - Unit testing Sp
  - ♦ 5 Java unit tes
  - JUnit with Hamc
  - Spring Boot in ur
- Other Interview Q&A f
  - Finance Domain In
    - 12+ FX or Forex
    - 15 Banking & fin
  - FIX Interview Q&A
    - 20+ FIX basics in
    - Finding your way
  - Groovy Interview Q
    - Groovy Coding C
      - Cash balance
      - Sum grades G
    - ♥ Q1 – Q5 Groov
    - ♦ 20 Groovy clos
    - ♦ 9 Groovy meta
    - Groovy method c

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

Senior Java developers must have a good handle on

open all | close all
- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience I
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction Managen

80+ step by step Java Tutorials

open all | close all
- Setting up Tutorial (6)
- Tutorial - Diagnosis (2
- Akka Tutorial (9)
- Core Java Tutorials (2
- Hadoop & Spark Tuto
- JEE Tutorials (19)
- Scala Tutorials (1)
- Spring & HIbernate Tu
- Tools Tutorials (19)
- Other Tutorials (45)

Preparing for Java written & coding tests

open all | close all
- ♦ Complete the given
- Can you write code? (
- Converting from A to I
- Designing your classe
- Java Data Structures
- Passing the unit tests
- What is wrong with th
- Writing Code Home A
- Written Test Core Jav
- Written Test JEE (1)

How good are your…to go places?

open all | close all
- Career Making Know-
- Job Hunting & Resum

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed