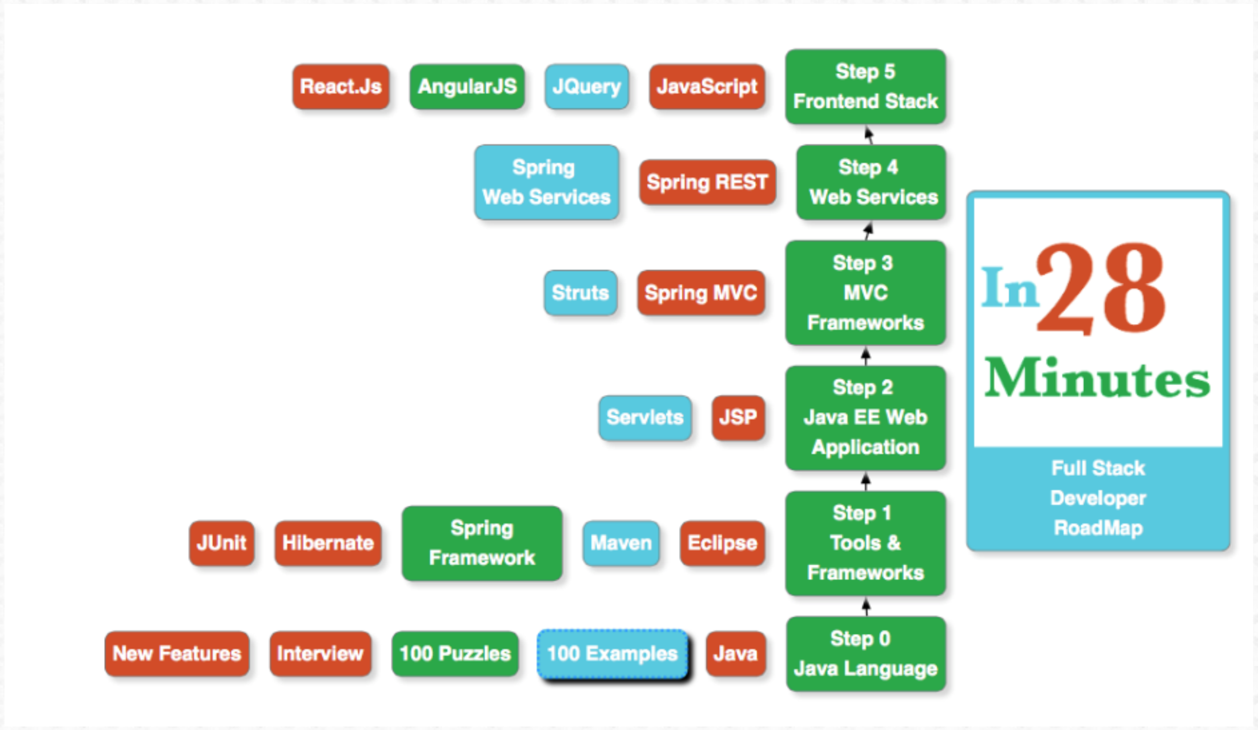


In28Minutes Course Guide

PROGRAMMING MADE EASY

25 Courses

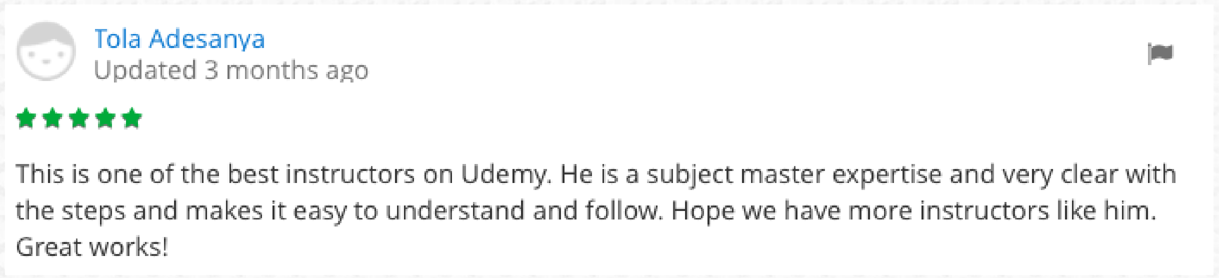
2 Million views

50,000 students

2000 5-STAR Reviews

10+ Free Courses

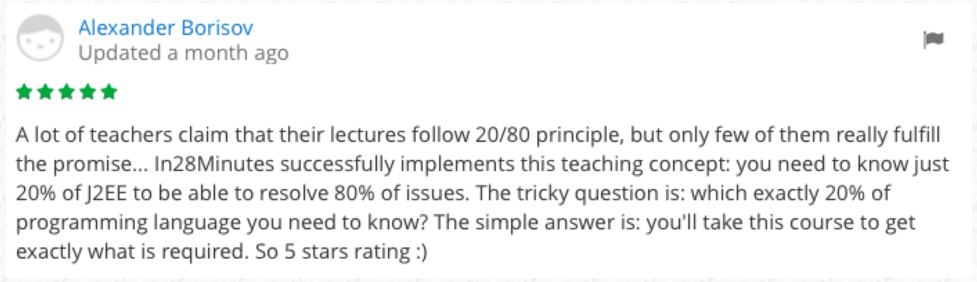
www.in28minutes.com



Tola Adesanya
Updated 3 months ago

★★★★★

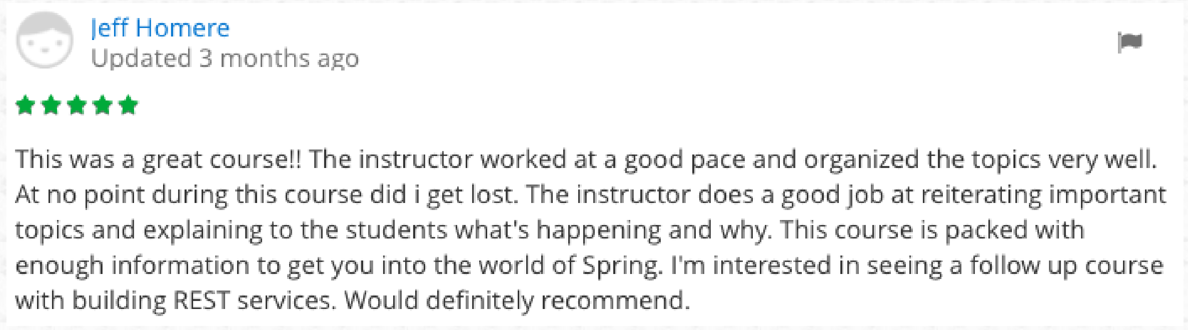
This is one of the best instructors on Udemy. He is a subject master expertise and very clear with the steps and makes it easy to understand and follow. Hope we have more instructors like him. Great works!



Alexander Borisov
Updated a month ago

★★★★★

A lot of teachers claim that their lectures follow 20/80 principle, but only few of them really fulfill the promise... In28Minutes successfully implements this teaching concept: you need to know just 20% of J2EE to be able to resolve 80% of issues. The tricky question is: which exactly 20% of programming language you need to know? The simple answer is: you'll take this course to get exactly what is required. So 5 stars rating :)



Jeff Homere
Updated 3 months ago

★★★★★

This was a great course!! The instructor worked at a good pace and organized the topics very well. At no point during this course did i get lost. The instructor does a good job at reiterating important topics and explaining to the students what's happening and why. This course is packed with enough information to get you into the world of Spring. I'm interested in seeing a follow up course with building REST services. Would definitely recommend.

OFFERS ON UDEMY COURSES

Java EE Design Patterns	SPECIAL OFFER
Spring MVC in 25 Steps	SPECIAL OFFER
JSP Servlets – 25 Steps	SPECIAL OFFER
Java Interview Guide	SPECIAL OFFER
Java OOPS Concepts	SPECIAL OFFER
Mockito – with 25 JUnit Examples	SPECIAL OFFER
Maven	SPECIAL OFFER

Java Interview Guide : 200+ Interview Questions and Answers

A Beginners Guide to Java Interview with 200+ Java Interview Questions

★★★★½ 4.5 (147 ratings) • 3,293 students enrolled

Instructed by in28Minutes Official   IT & Software / Other

Black Friday Deal

$10 ~~$60~~ 83% off

Lectures	**53**
Length	**6 hours**

Mirko Cukich
posted 10 days ago
★★★★★
Great material. The instructor is very clear and easy to understand. He knows the material very well. A++

Alvin Malate
posted a month ago
★★★★★
i learned something new even if ive been using java for a long time already

---

Navaneeth Kumar
posted a month ago
★★★★★
Excellent course.. Feeling very confident for attending Java interviews.. :)

GET IT NOW! $10 DISCOUNT COUPON

Java EE Made Easy - Patterns, Architecture and Frameworks

Beginners Guide to Demystifying Architecture, Patterns and Technologies used in Java EE. Get the Big Picture of Java EE.

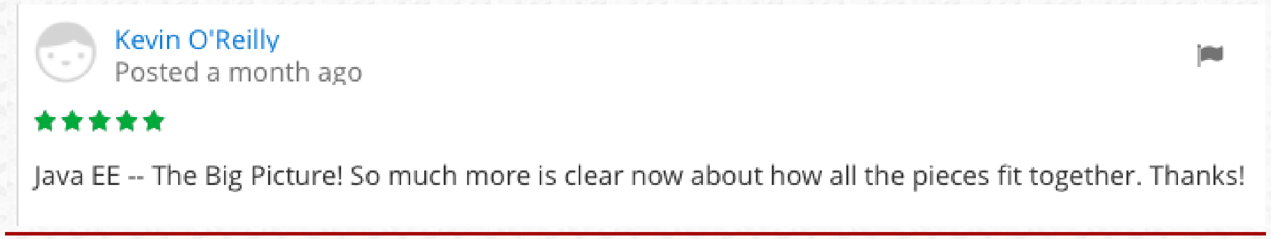
★★★★☆ 4.3 (180 ratings) • 3,992 students enrolled

Instructed by in28Minutes Official IT & Software / Other

Black Friday Deal

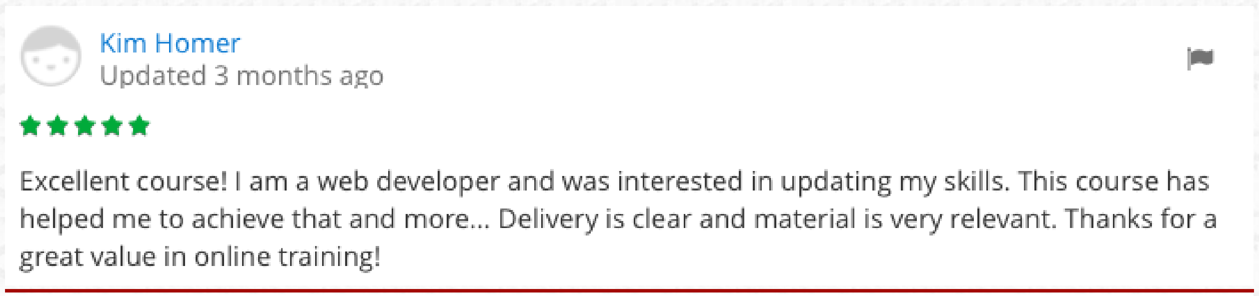
$10 ~~$40~~ 75% off

Lectures	**32**
Length	**2 hours**


Kevin O'Reilly
Posted a month ago

★★★★★

Java EE -- The Big Picture! So much more is clear now about how all the pieces fit together. Thanks!


Kim Homer
Updated 3 months ago

★★★★★

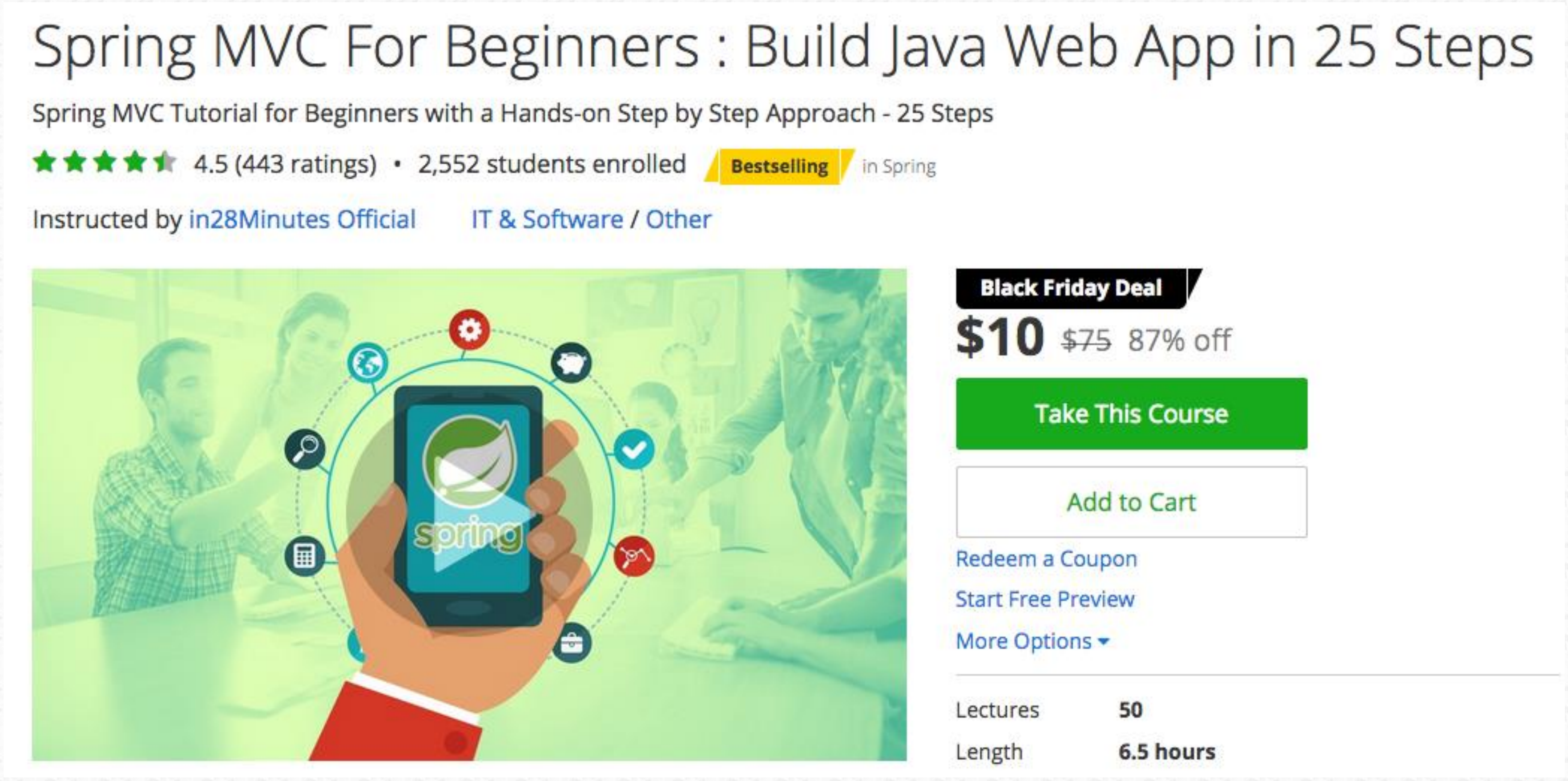
Excellent course! I am a web developer and was interested in updating my skills. This course has helped me to achieve that and more... Delivery is clear and material is very relevant. Thanks for a great value in online training!

Rahul S
Posted 3 months ago

★★★★★

This is an Excellent introduction to JavaEE. This helped me in learning a log of concepts in the was javaEE space. This a must course for every java web developer. All the concepts are explained in simple and very easy to understand terms. Thanks a lot for creating this unique course.

GET IT NOW! $10 DISCOUNT COUPON



Spring MVC For Beginners : Build Java Web App in 25 Steps

Spring MVC Tutorial for Beginners with a Hands-on Step by Step Approach - 25 Steps

★★★★☆ 4.5 (443 ratings) • 2,552 students enrolled Bestselling in Spring

Instructed by in28Minutes Official IT & Software / Other

Black Friday Deal

$10 ~~$75~~ 87% off

Take This Course

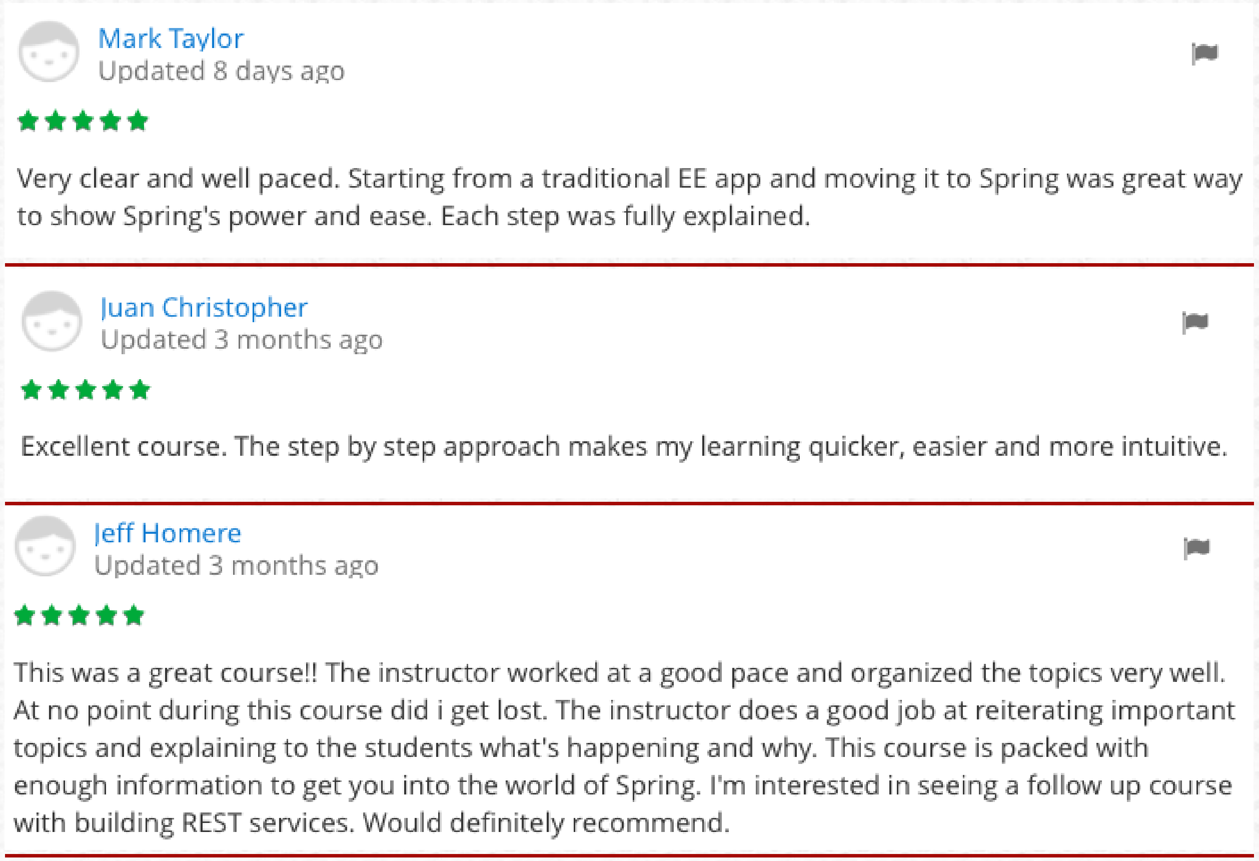
Add to Cart

Redeem a Coupon

Start Free Preview

More Options

Lectures	50
Length	6.5 hours



Mark Taylor
Updated 8 days ago

★★★★★

Very clear and well paced. Starting from a traditional EE app and moving it to Spring was great way to show Spring's power and ease. Each step was fully explained.

Juan Christopher
Updated 3 months ago

★★★★★

Excellent course. The step by step approach makes my learning quicker, easier and more intuitive.

Jeff Homere
Updated 3 months ago

★★★★★

This was a great course!! The instructor worked at a good pace and organized the topics very well. At no point during this course did i get lost. The instructor does a good job at reiterating important topics and explaining to the students what's happening and why. This course is packed with enough information to get you into the world of Spring. I'm interested in seeing a follow up course with building REST services. Would definitely recommend.

GET IT NOW! $10 DISCOUNT COUPON

Java Servlets and JSP - Build Java EE(JEE) app in 25 Steps

Servlets and JSP Tutorial for Beginners

★★★★★ 4.6 (111 ratings) • 1,904 students enrolled Bestselling in JavaServer Pages

Instructed by in28Minutes Official IT & Software / Other

Black Friday Deal

$10 ~~$40~~ 75% off

Take This Course

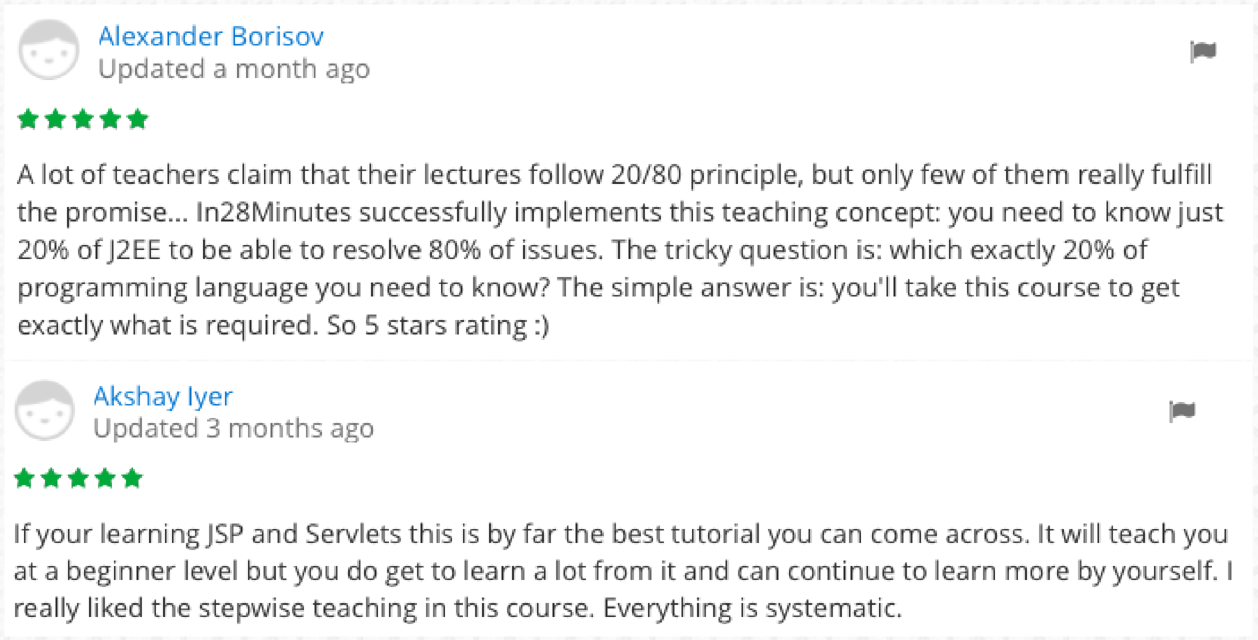
Add to Cart

Redeem a Coupon

Start Free Preview

More Options ▾

Lectures	35
Length	3.5 hours



Alexander Borisov
Updated a month ago

★★★★★

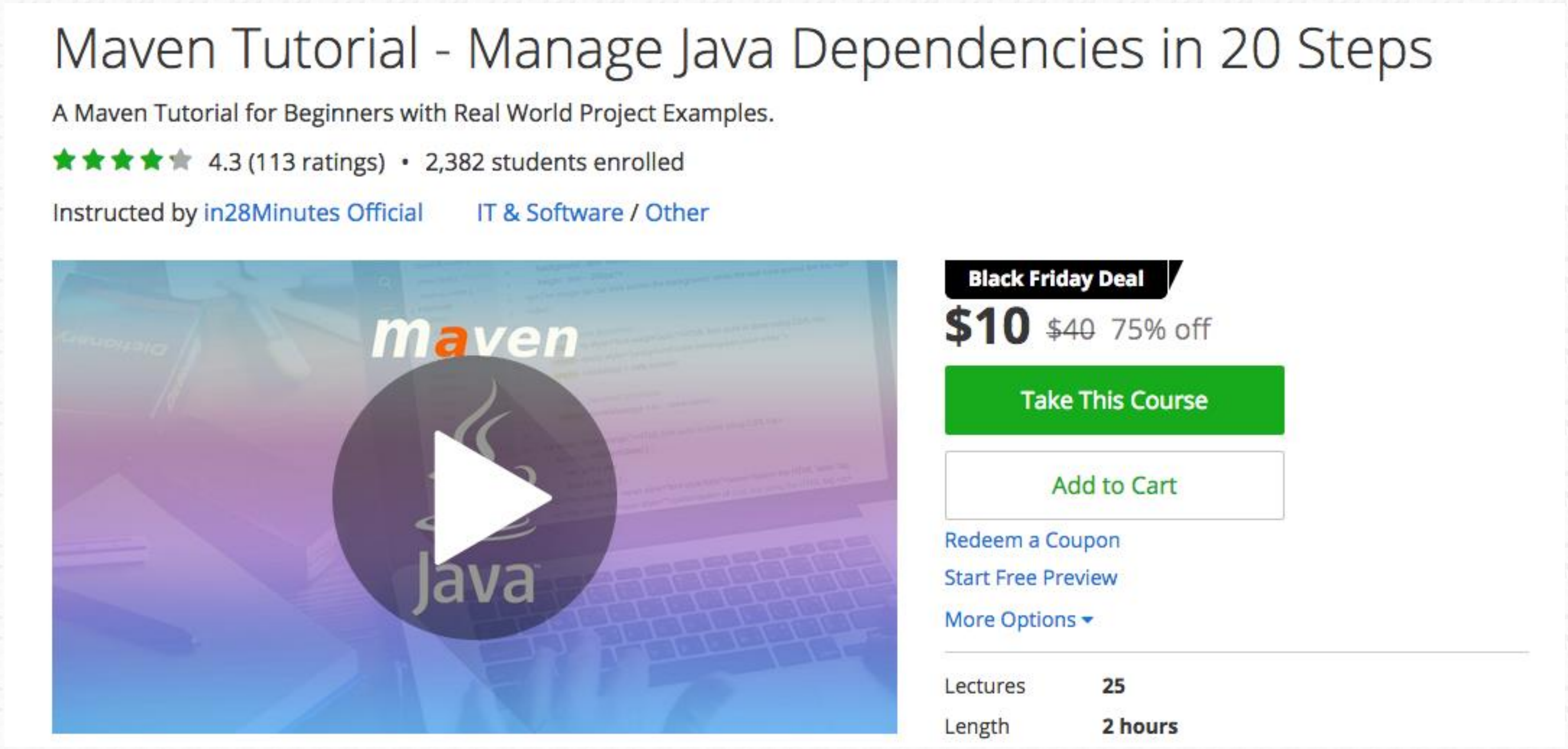
A lot of teachers claim that their lectures follow 20/80 principle, but only few of them really fulfill the promise... In28Minutes successfully implements this teaching concept: you need to know just 20% of J2EE to be able to resolve 80% of issues. The tricky question is: which exactly 20% of programming language you need to know? The simple answer is: you'll take this course to get exactly what is required. So 5 stars rating :)

Akshay Iyer
Updated 3 months ago

★★★★★

If your learning JSP and Servlets this is by far the best tutorial you can come across. It will teach you at a beginner level but you do get to learn a lot from it and can continue to learn more by yourself. I really liked the stepwise teaching in this course. Everything is systematic.

GET IT NOW! $10 DISCOUNT COUPON



Maven Tutorial - Manage Java Dependencies in 20 Steps

A Maven Tutorial for Beginners with Real World Project Examples.

★★★★☆ 4.3 (113 ratings) • 2,382 students enrolled

Instructed by in28Minutes Official IT & Software / Other

Black Friday Deal

$10 ~~$40~~ 75% off

Take This Course

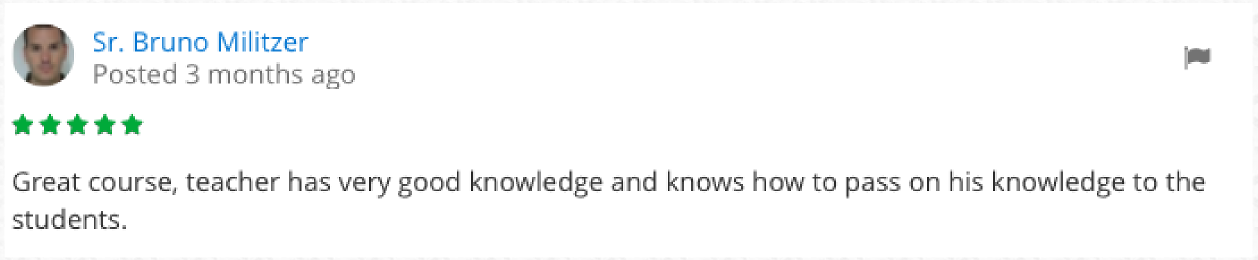
Add to Cart

Redeem a Coupon

Start Free Preview

More Options

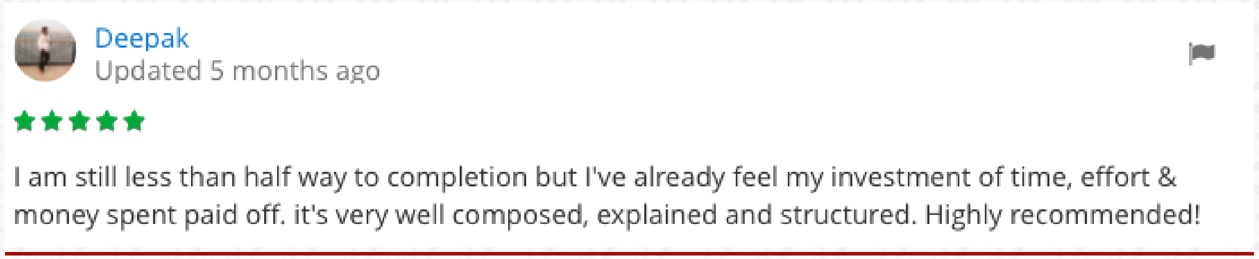
Lectures	25
Length	2 hours



Sr. Bruno Militzer
Posted 3 months ago

★★★★★

Great course, teacher has very good knowledge and knows how to pass on his knowledge to the students.

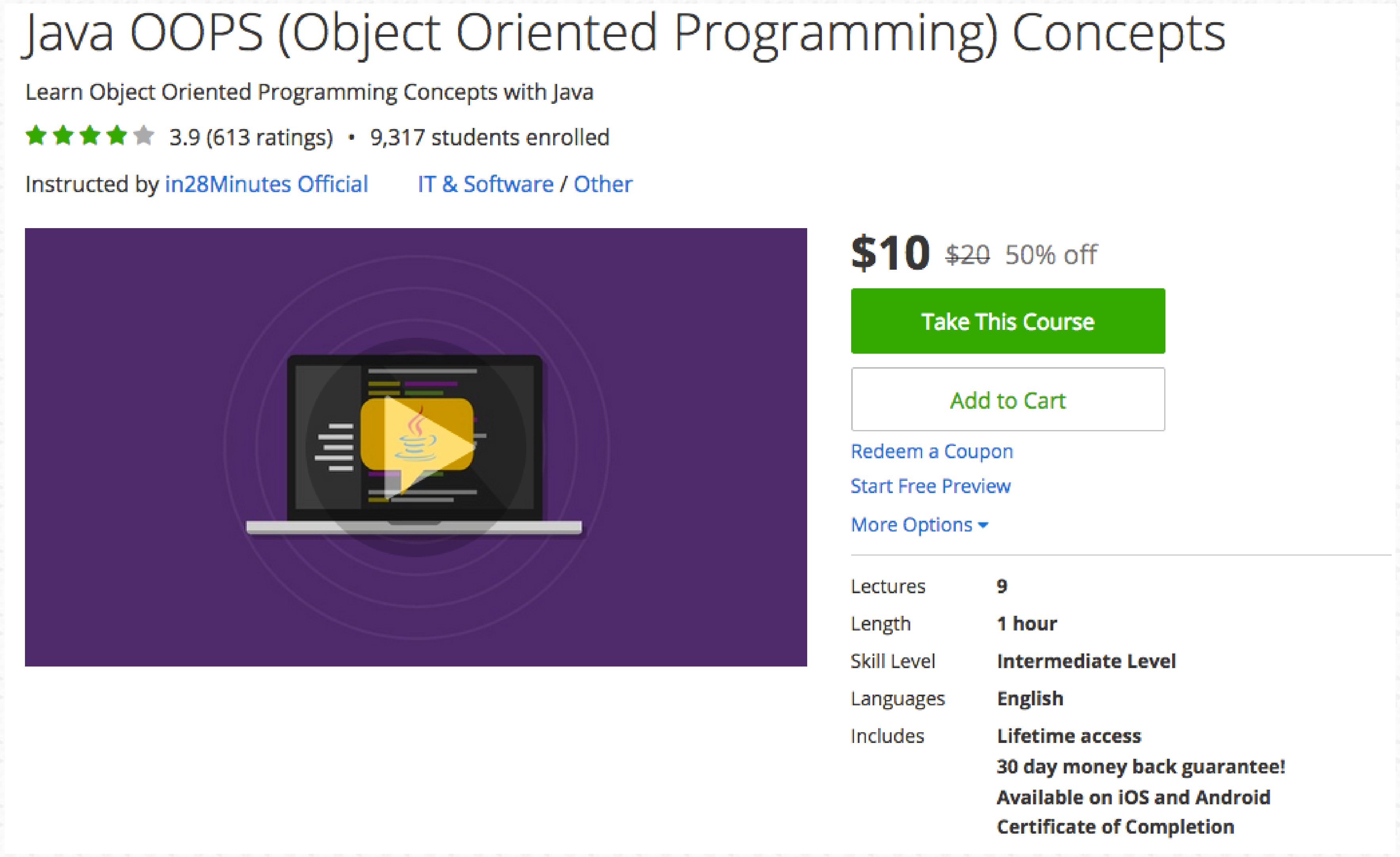


Deepak
Updated 5 months ago

★★★★★

I am still less than half way to completion but I've already feel my investment of time, effort & money spent paid off. it's very well composed, explained and structured. Highly recommended!

GET IT NOW! $10 DISCOUNT COUPON



Java OOPS (Object Oriented Programming) Concepts

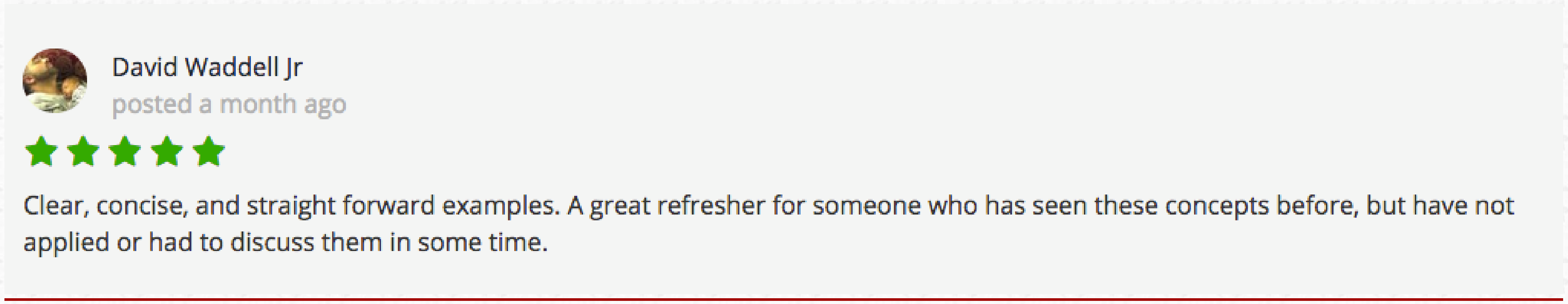
Learn Object Oriented Programming Concepts with Java

★★★★☆ 3.9 (613 ratings) • 9,317 students enrolled

Instructed by in28Minutes Official IT & Software / Other

$10 ~~$20~~ 50% off

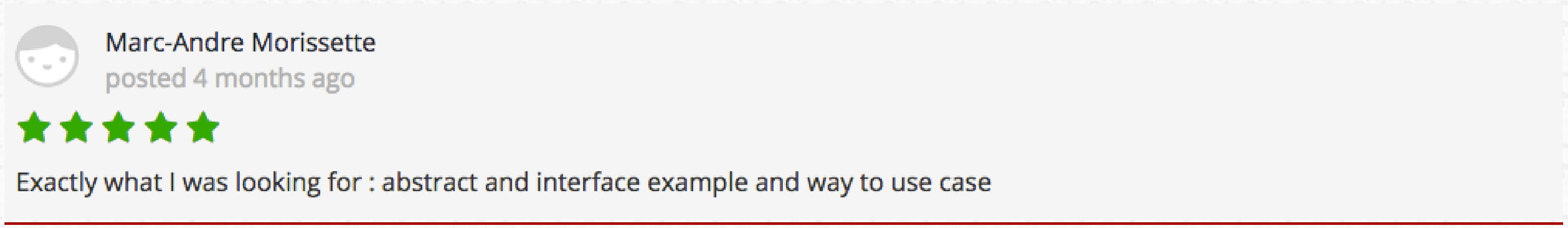
Lectures	**9**
Length	**1 hour**
Skill Level	**Intermediate Level**
Languages	**English**
Includes	**Lifetime access** **30 day money back guarantee!** **Available on iOS and Android** **Certificate of Completion**



David Waddell Jr
posted a month ago

★★★★★

Clear, concise, and straight forward examples. A great refresher for someone who has seen these concepts before, but have not applied or had to discuss them in some time.

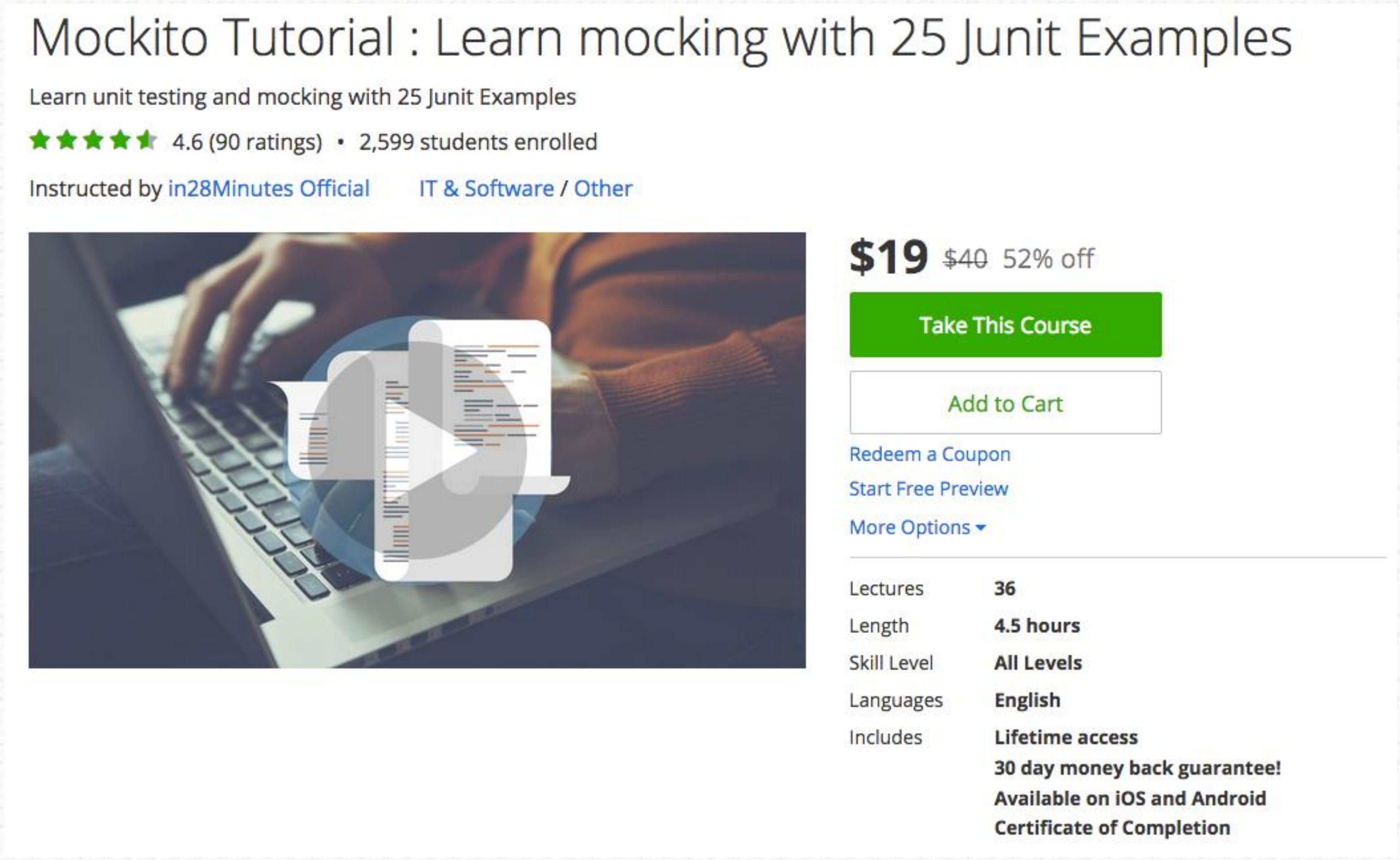


Marc-Andre Morissette
posted 4 months ago

★★★★★

Exactly what I was looking for : abstract and interface example and way to use case

GET IT NOW! $10 DISCOUNT COUPON



Mockito Tutorial : Learn mocking with 25 Junit Examples

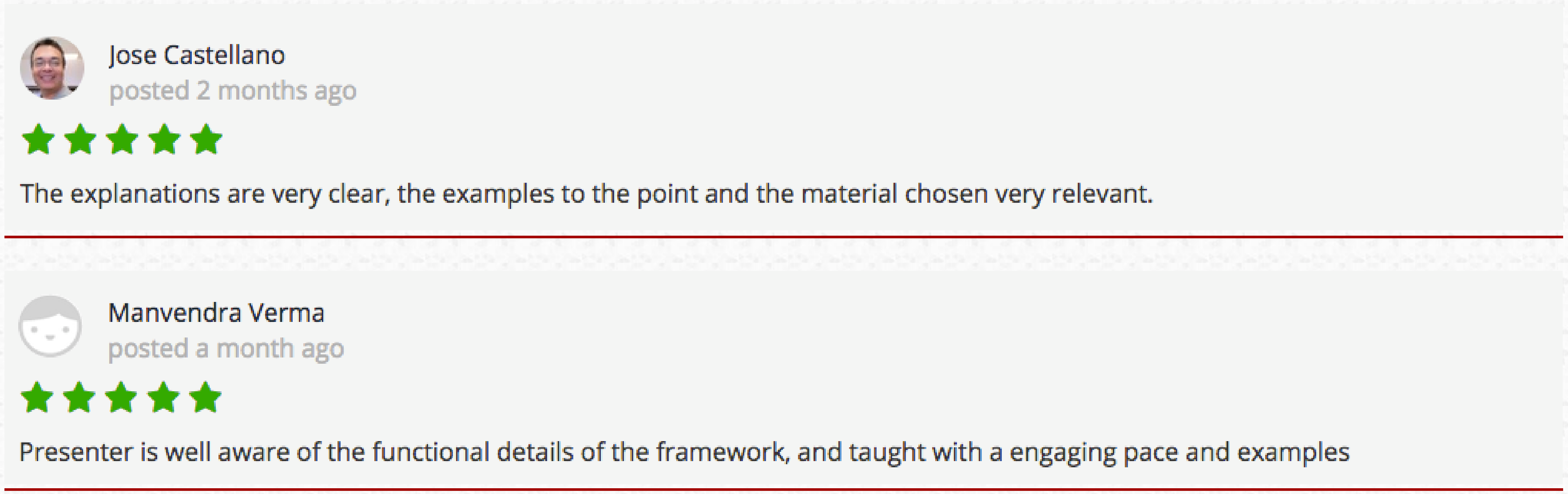
Learn unit testing and mocking with 25 Junit Examples

★★★★★ 4.6 (90 ratings) • 2,599 students enrolled

Instructed by in28Minutes Official   IT & Software / Other

$19 ~~$40~~ 52% off

Lectures	**36**
Length	**4.5 hours**
Skill Level	**All Levels**
Languages	**English**
Includes	**Lifetime access** **30 day money back guarantee!** **Available on iOS and Android** **Certificate of Completion**



Jose Castellano
posted 2 months ago

★★★★★

The explanations are very clear, the examples to the point and the material chosen very relevant.

Manvendra Verma
posted a month ago

★★★★★

Presenter is well aware of the functional details of the framework, and taught with a engaging pace and examples

GET IT NOW! $10 DISCOUNT COUPON

FREE COURSES

Eclipse Tutorial For Beginners : Learn Java IDE in

in28Minutes Official, Architect, Programmer and Trainer

★★★★☆ (317)

Free

LINK

A Beginner's Guide to Design Patterns

in28Minutes Official, Architect, Programmer and Trainer

★★★★☆ (144)

Free

LINK

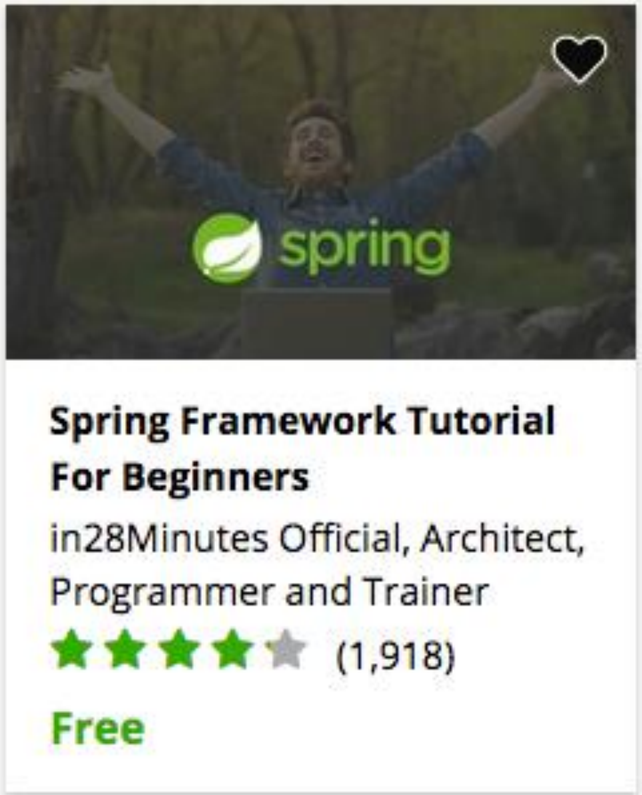
JUnit Tutorial for Beginners - Learn Java Unit Testing

in28Minutes Official, Architect, Programmer and Trainer

(1,365)

Free

LINK



Spring Framework Tutorial For Beginners

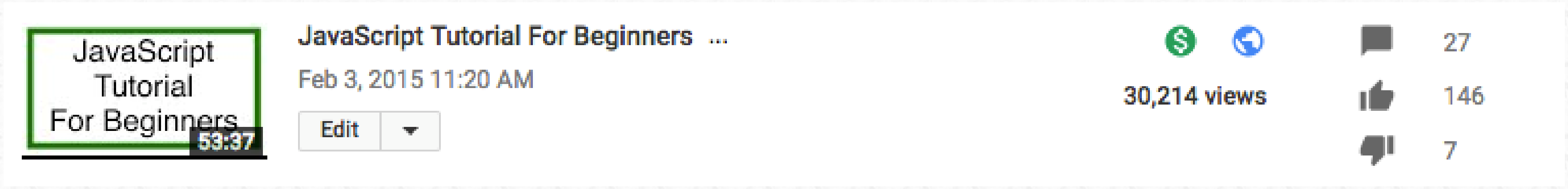
in28Minutes Official, Architect, Programmer and Trainer

(1,918)

Free

LINK

FREE YOUTUBE COURSES



JavaScript Tutorial For Beginners ...
Feb 3, 2015 11:20 AM
Edit
30,214 views
27
146
7

LINK

Introduction To Transaction Management HD
Mar 18, 2015 10:05 PM
Edit
29,255 views
32
125
8

LINK

Java Tutorial for Beginners
Jun 25, 2012 11:02 PM
Edit
23,695 views
5
49
3

LINK

Test Driven Development - Java Example 1

Jul 26, 2012 7:50 PM

21,894 views

11

183

7

LINK

Java Collection Framework 1 : Introduction to Collection Inteface ...

Mar 20, 2014 1:22 PM

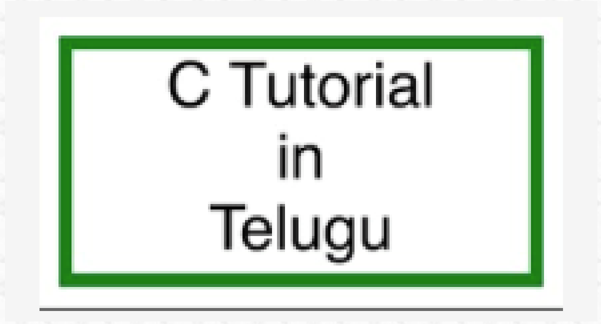
20,065 views

67

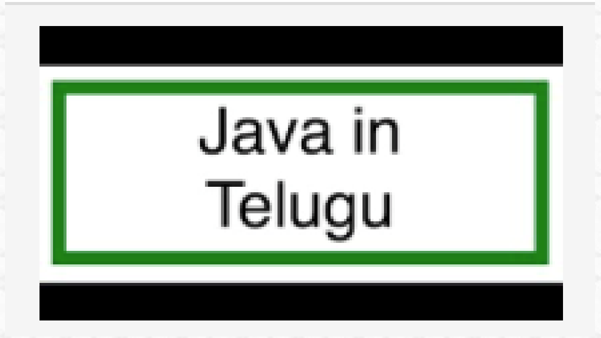
0

LINK

Couple of courses in my Mother Toungue - Telugu



https://www.youtube.com/watch?v=0Asyny8Wto4&list=PLA8E0AD777C0B9827



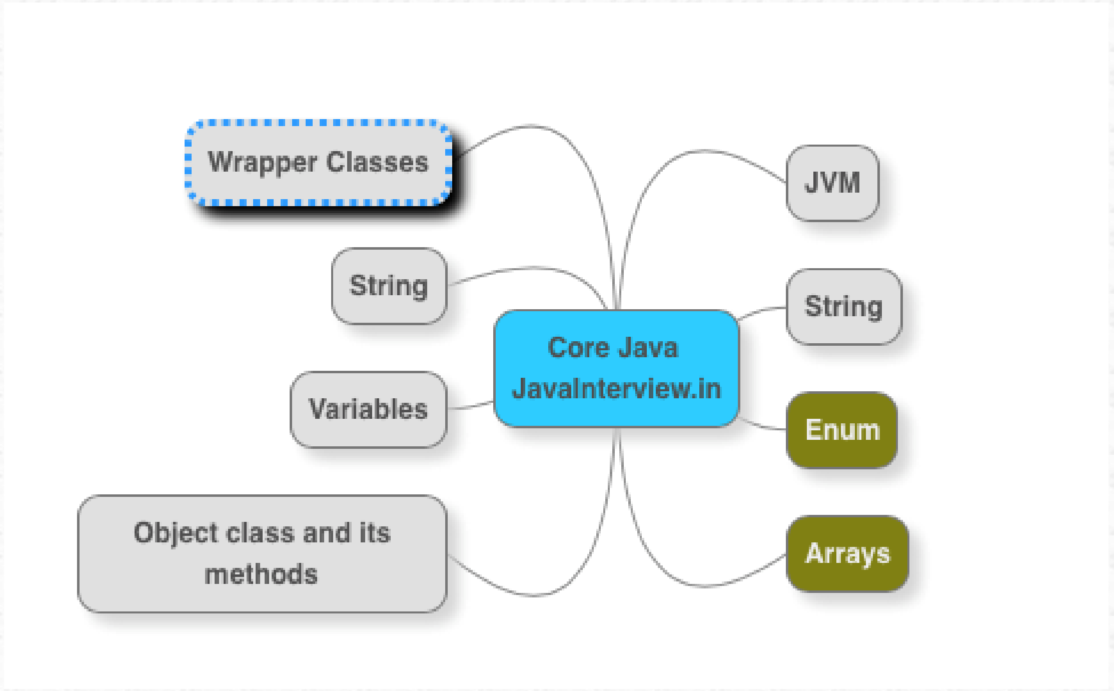
https://www.youtube.com/watch?v=xbxHQn1LjaM&list=PL5260E16312C8A27D

Java Interview Questions

Core Java

If you are an Experienced Programmer, Designer or Architect, we recommend to directly skip to the section on Advanced Java or Exception Handling or Threads and Synchronization or directly to the Frameworks (Spring, SpringMVC,Struts or Hibernate).

Following are the important topics that are important from interview perspective for core java.



Following videos cover these topics in great detail. In addition to following this guide, we recommend that you watch the videos as well.

Java Interview : A Freshers Guide - Part 1: https://www.youtube.com/watch?v=njZ48YVkei0.

Java Interview : A Freshers Guide - Part 2: https://www.youtube.com/watch?v=xyXuo0y-xoU

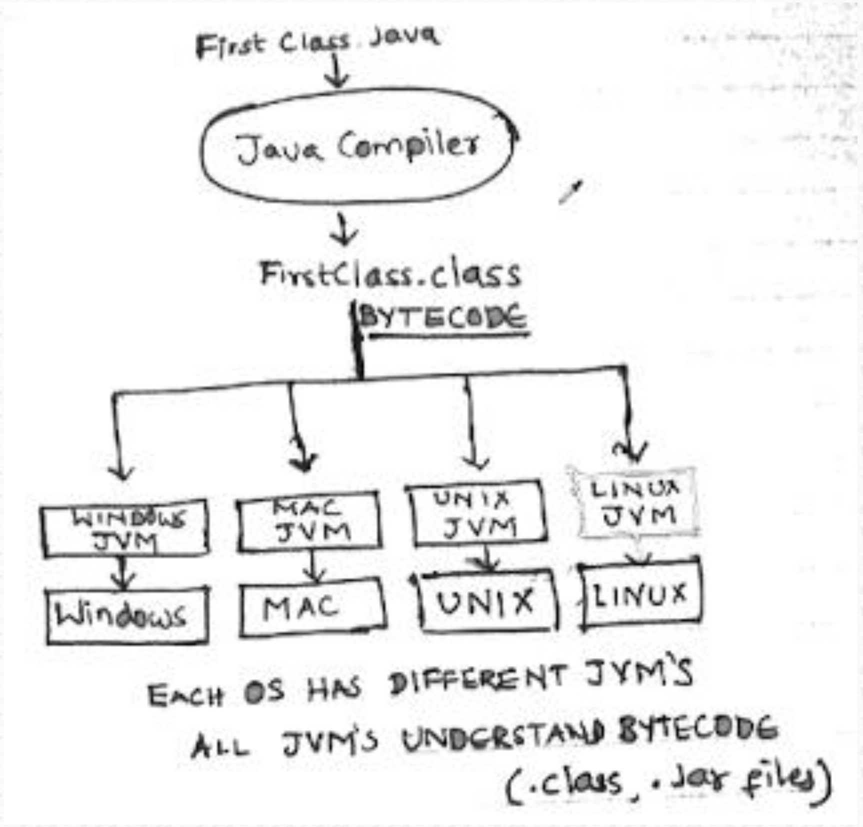
Why is Java so Popular?

Two main reasons for popularity of Java are

1. Platform Independence
2. Object Oriented Language

We will look at these in detail in later sections.

What is Platform Independence?



This video(https://www.youtube.com/watch?v=lLgcgvIHyAw) explains Platform Independence in great detail. Refer to it for more detailed answer.

Platform Independence is also called build once, run anywhere. Java is one of the most popular platform independent languages. Once we compile a java program and build a jar, we can run the jar (compiled java program) in any Operating System - where a JVM is installed.

Java achieves Platform Independence in a beautiful way. On compiling a java file the output is a class file - which contains an internal java representation called bytecode. JVM converts bytecode to executable instructions. The executable instructions are different in different operating systems. So, there are different JVM's for different operating systems. A JVM for windows is different from a JVM for mac. However, both the JVM's understand the bytecode and convert it to the executable code for the respective operating system.

What are the important differences between C++ and Java?

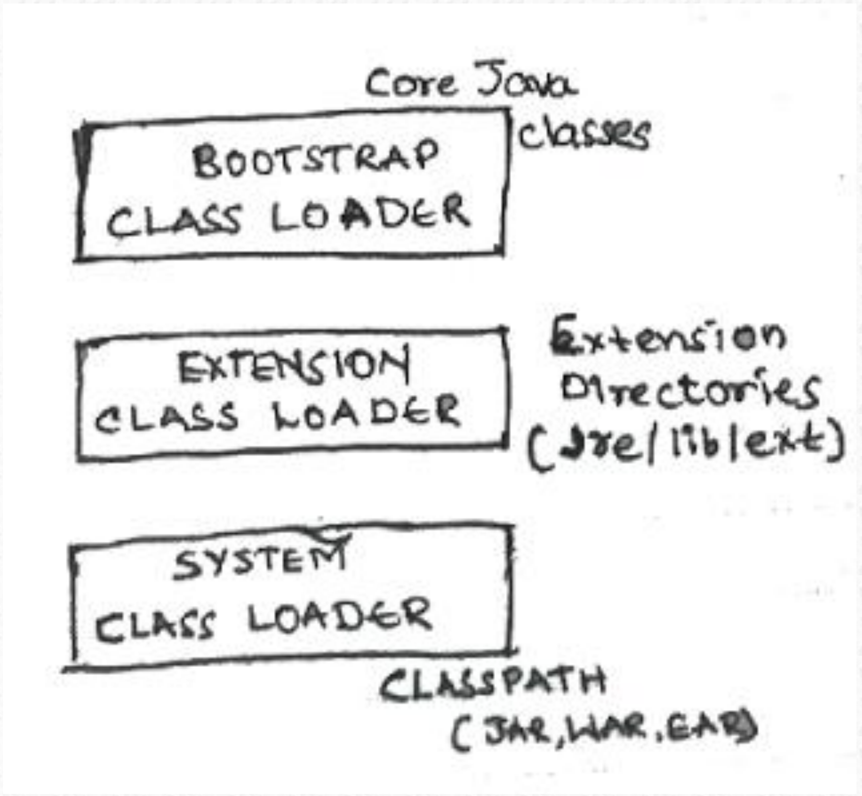
1. Java is platform independent. C++ is not platform independent.
2. Java is a pure Object Oriented Language (except for primitive variables). In C++, one can write structural programs without using objects.
3. C++ has pointers (access to internal memory). Java has no concept called pointers.
4. In C++, programmer has to handle memory management. A programmer has to write code to remove an object from memory. In Java, JVM takes care of removing objects from memory using a process called Garbage Collection.
5. C++ supports Multiple Inheritance. Java does not support Multiple Inheritance.

What is the role for a ClassLoader in Java?

A Java program is made up of a number of custom classes (written by programmers like us) and core classes (which come pre-packaged with Java). When a program is executed, JVM needs to load the content of all the needed class. JVM uses a ClassLoader to find the classes.

Three Class Loaders are shown in the picture

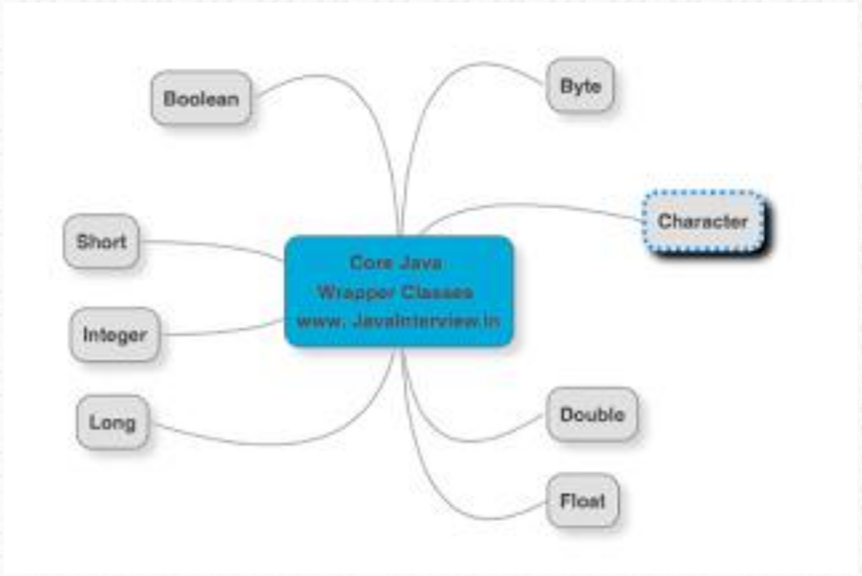
- System Class Loader - Loads all classes from CLASSPATH
- Extension Class Loader - Loads all classes from extension directory
- Bootstrap Class Loader - Loads all the Java core files



When JVM needs to find a class, it starts with System Class Loader. If it is not found, it checks with Extension Class Loader. If it not found, it goes to the Bootstrap Class Loader. If a class is still not found, a ClassNotFoundException is thrown.

What are wrapper classes?

This video(https://www.youtube.com/watch?v=YQbZRw2yIBk) covers the topic in great detail. A brief description is provided below.



A primitive wrapper class in the Java programming language is one of eight classes provided in the java.lang package to provide object methods for the eight primitive types. All of the primitive wrapper classes in Java are immutable.

Wrapper: Boolean,Byte,Character,Double,Float,Integer,Long,Short

Primitive: boolean,byte,char ,double, float, int , long,short

Wrapper classes are final and immutable. Examples of creating wrapper classes are listed below.

```
Integer number = new Integer(55);//int
Integer number2 = new Integer("55");//String
```

```
Float number3 = new Float(55.0);//double argument
Float number4 = new Float(55.0f);//float argument
Float number5 = new Float("55.0f");//String

Character c1 = new Character('C');//Only char constructor
//Character c2 = new Character(124);//COMPILER ERROR

Boolean b = new Boolean(true);

//"true" "True" "tRUe" - all String Values give True
//Anything else gives false
Boolean b1 = new Boolean("true");//value stored - true
Boolean b2 = new Boolean("True");//value stored - true
Boolean b3 = new Boolean("False");//value stored - false
Boolean b4 = new Boolean("SomeString");//value stored - false

b = false;
```

What are the different utility methods present in wrapper classes?

A number of utility methods are defined in wrapper classes to create and convert them from primitives.

valueOf Methods

Provide another way of creating a Wrapper Object

```
Integer seven =
    Integer.valueOf("111", 2);//binary 111 is converted to 7

Integer hundred =
    Integer.valueOf("100");//100 is stored in variable
```

xxxValue methods

xxxValue methods help in creating primitives

```
Integer integer = Integer.valueOf(57);
int primitive = seven.intValue();//57
float primitiveFloat = seven.floatValue();//57.0f

Float floatWrapper = Float.valueOf(57.0f);
int floatToInt = floatWrapper.intValue();//57
float floatToFloat = floatWrapper.floatValue();//57.0f
```

parseXxx methods

parseXxx methods are similar to valueOf but they return primitive values

```
int sevenPrimitive =
    Integer.parseInt("111", 2);//binary 111 is converted to 7

int hundredPrimitive =
    Integer.parseInt("100");//100 is stored in variable
```

static toString method

```
Look at the example of the toString static method below.
Integer wrapperEight = new Integer(8);
System.out.println(Integer.
        toString(wrapperEight));//String Output: 8
```

Overloaded static toString method

2nd parameter: radix

```
System.out.println(Integer

        .toString(wrapperEight, 2));//String Output: 1000
```

static toXxxString methods.

Xxx can be Hex,Binary,Octal

```
System.out.println(Integer
        .toHexString(wrapperEight));//String Output:8
System.out.println(Integer
        .toBinaryString(wrapperEight));//String Output:1000
System.out.println(Integer
        .toOctalString(wrapperEight));//String Output:10
```

What is Auto Boxing?

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

```
Integer ten = new Integer(10);
ten++;//allowed. Java does had work behind the screen for us
```

Boxing and new instances

Auto Boxing helps in saving memory by reusing already created Wrapper objects. However wrapper classes created using new are not reused.

Two wrapper objects created using new are not same object.

```
Integer nineA = new Integer(9);
Integer nineB = new Integer(9);
System.out.println(nineA == nineB);//false
System.out.println(nineA.equals(nineB));//true
```

Two wrapper objects created using boxing are same object.

```
Integer nineC = 9;
Integer nineD = 9;
System.out.println(nineC == nineD);//true
System.out.println(nineC.equals(nineD));//true
```

Are all String's immutable?

This video (https://www.youtube.com/watch?v=wh6L8zO_Hr4) covers all the topics related to String's in great detail. Refer to it for more details.

Value of a String Object once created cannot be modified. Any modification on a String object creates a new String object.

```
String str3 = "value1";
str3.concat("value2");
System.out.println(str3); //value1
```

Note that the value of str3 is not modified in the above example. The result should be assigned to a new reference variable (or same variable can be reused).

```
String concat = str3.concat("value2");
System.out.println(concat); //value1value2
```

Where are string literals stored in memory?

All strings literals are stored in "String constant pool". If compiler finds a String literal,it checks if it exists in the pool. If it exists, it is reused.

Following statement creates 1 string object (created on the pool) and 1 reference variable.

```
String str1 = "value";
```

However, if new operator is used to create string object, the new object is created on the heap.

Following piece of code create 2 objects.

```
//1. String Literal "value" - created in the "String constant pool"
//2. String Object - created on the heap
String str2 = new String("value");
```

Can you give examples of different utility methods in String class?

String class defines a number of methods to get information about the string content.

```
String str = "abcdefghijk";
```

Get information from String

```
Following methods help to get information from a String.
//char charAt(int paramInt)
System.out.println(str.charAt(2)); //prints a char - c
System.out.println("ABCDEFGH".length());//8
System.out.println("abcdefghij".toString()); //abcdefghij
System.out.println("ABC".equalsIgnoreCase("abc"));//true

//Get All characters from index paramInt
//String substring(int paramInt)
System.out.println("abcdefghij".substring(3)); //cdefghij
```

```
//All characters from index 3 to 6
System.out.println("abcdefghij".substring(3,7)); //defg
```

Explain about toString method in Java?

This video (https://www.youtube.com/watch?v=k02nM5ukV7w) covers toString in great detail. toString method is used to print the content of an Object. If the toString method is not overridden in a class, the default toString method from Object class is invoked. This would print some hashcode as shown in the example below. However, if toString method is overridden, the content returned by the toString method is printed.

Consider the class given below:

```
class Animal {

    public Animal(String name, String type) {
        this.name = name;
        this.type = type;
    }

    String name;
    String type;

}
```

Run this piece of code:

```
Animal animal = new Animal("Tommy", "Dog");
System.out.println(animal);//com.rithus.Animal@f7e6a96
```

Output does NOT show the content of animal (what name? and what type?). To show the content of the animal object, we can override the default implementation of toString method provided by Object class.

Adding toString to Animal class

```
class Animal {

    public Animal(String name, String type) {
        this.name = name;
        this.type = type;
    }

    String name;
    String type;

    public String toString() {
        return "Animal [name=" + name + ", type=" + type
                + "]";
    }

}
```

Run this piece of code:

```
Animal animal = new Animal("Tommy","Dog");
System.out.println(animal);//Animal [name=Tommy, type=Dog]
```

Output now shows the content of the animal object.

What is the use of equals method in Java?

Equals method is used when we compare two objects. Default implementation of equals method is defined in Object class. The implementation is similar to == operator. Two object references are equal only if they are pointing to the same object.

We need to override equals method, if we would want to compare the contents of an object.

Consider the example Client class provided below.

```java
class Client {
    private int id;

    public Client(int id) {
        this.id = id;
    }
}
```

== comparison operator checks if the object references are pointing to the same object. It does NOT look at the content of the object.

```java
Client client1 = new Client(25);
Client client2 = new Client(25);
Client client3 = client1;

//client1 and client2 are pointing to different client objects.
System.out.println(client1 == client2);//false
```

```
//client3 and client1 refer to the same client objects.
System.out.println(client1 == client3);//true

//similar output to ==
System.out.println(client1.equals(client2));//false
System.out.println(client1.equals(client3));//true
```

We can override the equals method in the Client class to check the content of the objects. Consider the example below: The implementation of equals method checks if the id's of both objects are equal. If so, it returns true. Note that this is a basic implementation of equals and more needs to be done to make it fool-proof.

```
class Client {
    private int id;

    public Client(int id) {
        this.id = id;
    }

    @Override
    public boolean equals(Object obj) {
        Client other = (Client) obj;
        if (id != other.id)
            return false;
        return true;
    }
}
```

Consider running the code below:

```
Client client1 = new Client(25);
Client client2 = new Client(25);
Client client3 = client1;

//both id's are 25
System.out.println(client1.equals(client2));//true

//both id's are 25
System.out.println(client1.equals(client3));//true
```

Above code compares the values (id's) of the objects.

What are the important things to consider when implementing equals method?

Any equals implementation should satisfy these properties:

- Reflexive. For any reference value x, x.equals(x) returns true.
- Symmetric. For any reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- Transitive. For any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) must return true.
- Consistent. For any reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, if no information used in equals is modified.
- For any non-null reference value x, x.equals(null) should return false.

Let's now provide an implementation of equals which satisfy these properties:

```
//Client class
@Override
```

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Client other = (Client) obj;
    if (id != other.id)
        return false;
    return true;
}
```

What is the hashCode method used for in Java?

HashCode's are used in hashing to decide which group (or bucket) an object should be placed into. A group of object's might share the same hashcode.

The implementation of hash code decides effectiveness of Hashing. A good hashing function evenly distributes object's into different groups (or buckets).

A good hashCode method should have the following properties

- If obj1.equals(obj2) is true, then obj1.hashCode() should be equal to obj2.hashCode()
- obj.hashCode() should return the same value when run multiple times, if values of obj used in equals() have not changed.
- If obj1.equals(obj2) is false, it is NOT required that obj1.hashCode() is not equal to obj2.hashCode(). Two unequal objects MIGHT have the same hashCode.

A sample hashcode implementation of Client class which meets above constraints is given below:

```
//Client class
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + id;
    return result;
}
```

What is casting?

Casting is used when we want to convert on data type to another.

There are two types of Casting

- Implicit Casting
- Explicit Casting

What is Implicit Casting?

Implicit Casting is done by the compiler. Good examples of implicit casting are all the automatic widening conversions i.e. storing smaller values in larger variable types.

```
int value = 100;
long number = value; //Implicit Casting
float f = 100; //Implicit Casting
```

What is Explicit Casting?

Explicit Casting is done through code. Good examples of explicit casting are the narrowing conversions. Storing larger values into smaller variable types;

```
long number1 = 25678;
int number2 = (int)number1;//Explicit Casting
//int x = 35.35;//COMPILER ERROR
int x = (int)35.35;//Explicit Casting
```

Explicit casting would cause truncation of value if the value stored is greater than the size of the variable.

```
int bigValue = 280;
byte small = (byte) bigValue;
System.out.println(small);//output 24. Only 8 bits remain.
```

How are variables initialized in Java?

Member and Static variables are alway initialized with default values. Default values for numeric types is 0, floating point types is 0.0, boolean is false, char is '\u0000' and object reference variable is null.

Local/block variables are NOT initialized by compiler.

If local variables are used before initialization, it would result in a compilation error.

```
package com.rithus.variables;

public class VariableInitialization {
    public static void main(String[] args) {
        Player player = new Player();

        //score is an int member variable - default 0
        System.out.println(player.score);//0 - RULE1

        //name is a member reference variable - default null
        System.out.println(player.name);//null - RULE1

        int local; //not initialized
        //System.out.println(local);//COMPILER ERROR! RULE3

        String value1;//not initialized
        //System.out.println(value1);//COMPILER ERROR! RULE3

        String value2 = null;//initialized
        System.out.println(value2);//null - NO PROBLEM.
    }
}
```

What is a nested if else? Can you explain with an example?

Look at the example below. The code in first if condition which is true is executed. If none of the if conditions are true, then code in else is executed.

```
int z = 15;
if(z==10){
    System.out.println("Z is 10");//NOT executed
} else if(z==12){
    System.out.println("Z is 12");//NOT executed
} else if(z==15){
    System.out.println("Z is 15");//executed. Rest of the if else are skipped.
} else {
    System.out.println("Z is Something Else.");//NOT executed
}

z = 18;
if(z==10){
    System.out.println("Z is 10");//NOT executed
} else if(z==12){
    System.out.println("Z is 12");//NOT executed
} else if(z==15){
    System.out.println("Z is 15");//NOT executed
} else {
    System.out.println("Z is Something Else.");//executed
}
```

Arrays

Refer to this video(https://www.youtube.com/watch?v=8bVysCXT-io) for exhaustive coverage of all the interview questions about arrays.

How do you declare and create an array?

Let's first discuss about how to declare an array. All below ways are legal. However, using the third format of declaration is recommended.

```
int marks[]; //Not Readable
int[] runs; //Not Readable
int[] temperatures;//Recommended
```

Declaration of an Array should not include size.

```
//int values[5];//Compilation Error!
```

Declaring 2D Array Examples:

```
int[][] matrix1; //Recommended
int[] matrix2[]; //Legal but not readable. Avoid.
```

Lets now look at how to create an array (define a size and allocate memory).

```
marks = new int[5]; // 5 is size of array
```

Declaring and creating an array in same line.

```
int marks2[] = new int[5];
```

Can the size of an array be changed dynamically?

Once An Array is created, its size cannot be changed.

Can you create an array without defining size of an array?

Size of an array is mandatory to create an array.

```
//marks = new int[];//COMPILER ERROR
```

What are the default values in an array?

New Arrays are always initialized with default values.

```
int marks2[] = new int[5];
System.out.println(marks2[0]);//0
```

Default Values

byte,short,int,long	0
float,double	0.0
boolean	false
object	null

How do you loop around an array using enhanced for loop?

Name of the variable is mark and the array we want to loop around is marks.

```
for (int mark: marks) {
    System.out.println(mark);
}
```

How do you print the content of an array?

Let’s look at different methods in java to print the content of an array.

Printing a 1D Array

```
int marks5[] = { 25, 30, 50, 10, 5 };
System.out.println(marks5); //[I@6db3f829
System.out.println(
    Arrays.toString(marks5));//[25, 30, 50, 10, 5]
```

Printing a 2D Array

```
int[][] matrix3 = { { 1, 2, 3 }, { 4, 5, 6 } };
System.out.println(matrix3); //[[I@1d5a0305
System.out.println(
        Arrays.toString(matrix3));
//[[I@6db3f829, [I@42698403]
System.out.println(
        Arrays.deepToString(matrix3));
//[[1, 2, 3], [4, 5, 6]]
```

matrix3[0] is a 1D Array

```
System.out.println(matrix3[0]);//[I@86c347
System.out.println(
        Arrays.toString(matrix3[0]));//[1, 2, 3]
```

How do you compare two arrays?

Arrays can be compared using static method equals defined in Arrays class. Two arrays are equal only if they have the same numbers in all positions and have the same size.

```
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 4, 5, 6 };

System.out.println(Arrays
        .equals(numbers1, numbers2)); //false

int[] numbers3 = { 1, 2, 3 };
System.out.println(Arrays
        .equals(numbers1, numbers3)); //true
```

How do you sort an array?

Array's can be sorted using static utility method sort defined in Arrays class.

```
int rollNos[] = { 12, 5, 7, 9 };
Arrays.sort(rollNos);
System.out.println(
        Arrays.toString(rollNos));//[5, 7, 9, 12]
```

Enum

Refer to this video (https://www.youtube.com/watch?v=64Y7EP8-Ark) for exhaustive coverage of all the interview questions about Enum.

What is an Enum?

Enum allows specifying a list of values for a Type. Consider the example below. It declares an enum Season with 4 possible values.

```java
enum Season {
    WINTER, SPRING, SUMMER, FALL
};
```

How do you create an enum from a String value?

Function valueOf(String) is used to convert a string to enum.

```java
        //Converting String to Enum
        Season season = Season.valueOf("FALL");
```

Function name() is used to find String value of an enum.

```java
        //Converting Enum to String
        System.out.println(season.name());//FALL
```

What is an Enum Ordinal?

Java assigns default ordinals to an enum in order. However, it is not recommended to use ordinals to perform logic.

```java
        //Default ordinals of enum
        // By default java assigns ordinals in order
        System.out.println(Season.WINTER.ordinal());//0
        System.out.println(Season.SPRING.ordinal());//1
        System.out.println(Season.SUMMER.ordinal());//2
        System.out.println(Season.FALL.ordinal());//3
```

How do you compare two Enums?

Values of Enum’s can be compared using == or the equals function.

```
        //Comparing two Enums
        Season season1 = Season.FALL;
        Season season2 = Season.FALL;
        System.out.println(season1 == season2);//true
        System.out.println(season1.equals(season2));//true
```

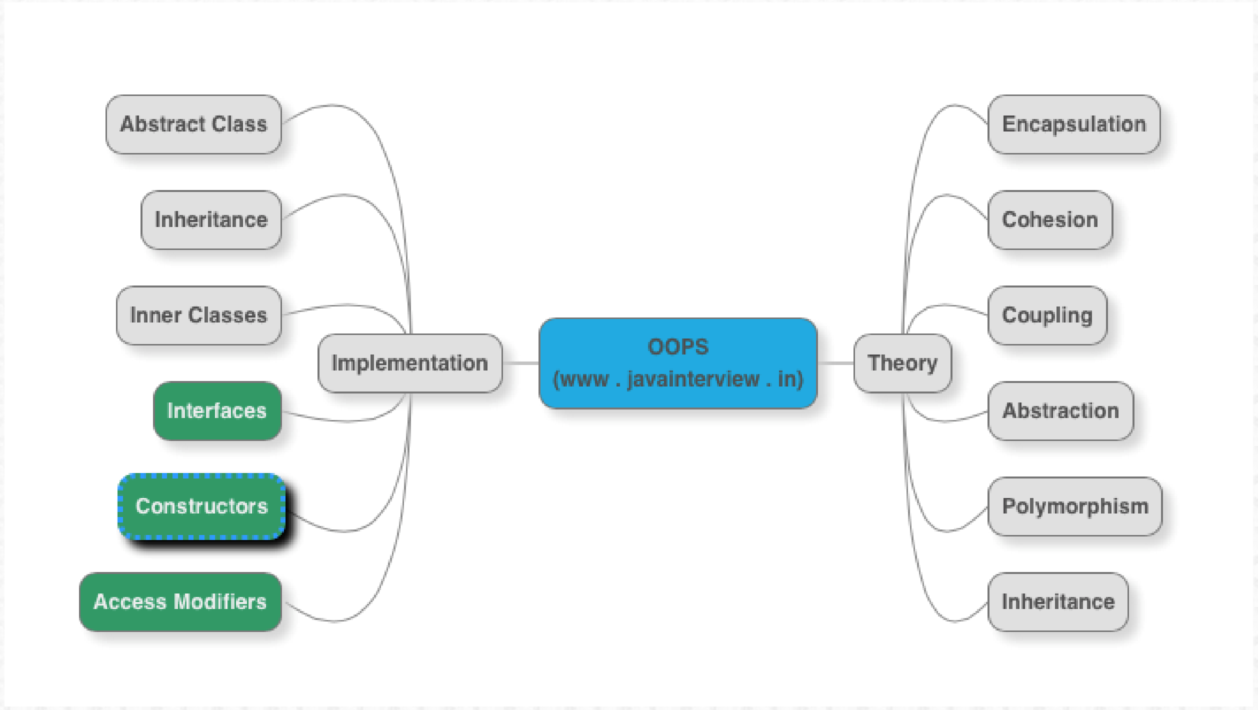
Can you use a Switch Statement around an Enum?

Example below shows how we can use a switch around an enum.

```
        //Using switch statement on an enum
        public int getExpectedMaxTemperature() {
            switch (this) {
            case WINTER:
                return 5;
            case SPRING:
            case FALL:
                return 10;
            case SUMMER:
                return 20;
            }
                 return -1;// Dummy since Java does not recognize this is possible            }
```

OOPS

Following picture show the topics we would cover in this article.



What is the super class of every class in Java?

Every class in java is a sub class of the class Object. When we create a class we inherit all the methods and properties of Object class. Let's look at a simple example:

```
String str = "Testing";
System.out.println(str.toString());
System.out.println(str.hashCode());
System.out.println(str.clone());
```

```java
if(str instanceof Object){
    System.out.println("I extend Object");//Will be printed
}
```

In the above example, toString, hashCode and clone methods for String class are inherited from Object class and overridden.

Can super class reference variable can hold an object of sub class?

Yes. Look at the example below:

Actor reference variables actor1, actor2 hold the reference of objects of sub classes of Animal, Comedian and Hero.

Since object is super class of all classes, an Object reference variable can also hold an instance of any class.

```java
//Object is super class of all java classes
Object object = new Hero();

public class Actor {
    public void act(){
        System.out.println("Act");
    };
}

//IS-A relationship. Hero is-a Actor
public class Hero extends Actor {
    public void fight(){
        System.out.println("fight");
    };
}

//IS-A relationship. Comedian is-a Actor
public class Comedian extends Actor {
    public void performComedy(){
        System.out.println("Comedy");
    };
}

Actor actor1 = new Comedian();
Actor actor2 = new Hero();
```

Is Multiple Inheritance allowed in Java?

Multiple Inheritance results in a number of complexities. Java does not support Multiple Inheritance.

```
class Dog extends Animal, Pet { //COMPILER ERROR
}
```

However, we can create an Inheritance Chain

```
class Pet extends Animal {
}

class Dog extends Pet {
}
```

What is Polymorphism?

Refer to this video(https://www.youtube.com/watch?v=t8PTatUXtpI) for a clear explanation of polymorphism.

Polymorphism is defined as “Same Code” giving “Different Behavior”. Let’s look at an example.

Let’s define an Animal class with a method shout.

```java
public class Animal {
    public String shout() {
        return "Don't Know!";
    }
}
```

Let’s create two new sub classes of Animal overriding the existing shout method in Animal.

```java
class Cat extends Animal {
    public String shout() {
        return "Meow Meow";
    }
}


class Dog extends Animal {
    public String shout() {
        return "BOW BOW";
    }

    public void run(){

    }
}
```

Look at the code below. An instance of Animal class is created. shout method is called.

```java
Animal animal1 = new Animal();
System.out.println(
        animal1.shout()); //Don't Know!
```

Look at the code below. An instance of Dog class is created and store in a reference variable of type Animal.

```
Animal animal2 = new Dog();

//Reference variable type => Animal
//Object referred to => Dog
//Dog's bark method is called.
System.out.println(
        animal2.shout()); //BOW BOW
```

When shout method is called on animal2, it invokes the shout method in Dog class (type of the object pointed to by reference variable animal2).

Even though dog has a method run, it cannot be invoked using super class reference variable.

```
//animal2.run();//COMPILE ERROR
```

What is the use of instanceof Operator in Java?

instanceof operator checks if an object is of a particular type. Let us consider the following class and interface declarations:

```
class SuperClass {
};

class SubClass extends SuperClass {
};
```

```
interface Interface {
};

class SuperClassImplementingInteface implements Interface {
};

class SubClass2 extends SuperClassImplementingInteface {
};

class SomeOtherClass {
};
```

Let’s consider the code below. We create a few instances of the classes declared above.

```
SubClass subClass = new SubClass();
Object subClassObj = new SubClass();

SubClass2 subClass2 = new SubClass2();
SomeOtherClass someOtherClass = new SomeOtherClass();
```

Let’s now run instanceof operator on the different instances created earlier.

```
System.out.println(subClass instanceof SubClass);//true
System.out.println(subClass instanceof SuperClass);//true
System.out.println(subClassObj instanceof SuperClass);//true

System.out.println(subClass2
        instanceof SuperClassImplementingInteface);//true
```

instanceof can be used with interfaces as well. Since Super Class implements the interface, below code prints true.

```
System.out.println(subClass2
        instanceof Interface);//true
```

If the type compared is unrelated to the object, a compilation error occurs.

```
//System.out.println(subClass
//            instanceof SomeOtherClass);//Compiler Error
```

Object referred by subClassObj(SubClass)- NOT of type SomeOtherClass

```
System.out.println(subClassObj instanceof SomeOtherClass);//false
```

What is an Abstract Class?

An abstract class (Video Link - https://www.youtube.com/watch?v=j3GLUcdlz1w) is a class that cannot be instantiated, but must be inherited from. An abstract class may be fully implemented, but is more usually partially implemented or not implemented at all, thereby encapsulating common functionality for inherited classes.

In code below "AbstractClassExample ex = new AbstractClassExample();" gives a compilation error because AbstractClassExample is declared with keyword abstract.

```
public abstract class AbstractClassExample {
    public static void main(String[] args) {
        //An abstract class cannot be instantiated
        //Below line gives compilation error if uncommented
        //AbstractClassExample ex = new AbstractClassExample();
    }
}
```

How do you define an abstract method?

An Abstract method does not contain body. An abstract method does not have any implementation. The implementation of an abstract method should be provided in an over-riding method in a sub class.

```
    //Abstract Class can contain 0 or more abstract methods
    //Abstract method does not have a body
    abstract void abstractMethod1();
    abstract void abstractMethod2();
```

Abstract method can be declared only in Abstract Class. In the example below, abstractMethod() gives a compiler error because NormalClass is not abstract.

```
class NormalClass{
    abstract void abstractMethod();//COMPILER ERROR
}
```

What is Coupling?

Coupling is a measure of how much a class is dependent on other classes. There should minimal dependencies between classes. So, we should always aim for low coupling between classes.

Coupling Example Problem

Consider the example below:

```
class ShoppingCartEntry {
    public float price;
    public int quantity;
}

class ShoppingCart {
    public ShoppingCartEntry[] items;
}

class Order {
    private ShoppingCart cart;
    private float salesTax;

    public Order(ShoppingCart cart, float salesTax) {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    // This method know the internal details of ShoppingCartEntry and
    // ShoppingCart classes. If there is any change in any of those
    // classes, this method also needs to change.
    public float orderTotalPrice() {
        float cartTotalPrice = 0;
        for (int i = 0; i < cart.items.length; i++) {
            cartTotalPrice += cart.items[i].price
                    * cart.items[i].quantity;
        }
        cartTotalPrice += cartTotalPrice * salesTax;
        return cartTotalPrice;
    }
}
```

Method orderTotalPrice in Order class is coupled heavily with `ShoppingCartEntry` and `ShoppingCart` classes. It uses different properties (items, price, quantity) from these classes. If any of these properties change, orderTotalPrice will also change. This is not good for Maintenance.

Solution

Consider a better implementation with lesser coupling between classes below: In this implementation, changes in ShoppingCartEntry or CartContents might not affect Order class at all.

```
class ShoppingCartEntry
{
    float price;
    int quantity;

    public float getTotalPrice()
    {
        return price * quantity;
    }
}

class CartContents
{
    ShoppingCartEntry[] items;

    public float getTotalPrice()
    {
        float totalPrice = 0;
        for (ShoppingCartEntry item:items)
        {
            totalPrice += item.getTotalPrice();
        }
        return totalPrice;
    }
}

class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float totalPrice()
    {
        return cart.getTotalPrice() * (1.0f + salesTax);
    }
}
```

What is Cohesion?

Cohesion (Video Link - https://www.youtube.com/watch?v=BkcQWoF5124) is a measure of how related the responsibilities of a class are. A class must be highly cohesive i.e. its responsibilities (methods) should be highly related to one another.

Example Problem

Example class below is downloading from internet, parsing data and storing data to database. The responsibilities of this class are not really related. This is not cohesive class.

```
class DownloadAndStore{
    void downloadFromInternet(){
    }

    void parseData(){
    }
    void storeIntoDatabase(){
    }

    void doEverything(){
        downloadFromInternet();
        parseData();
        storeIntoDatabase();
    }
}
```

Solution

This is a better way of approaching the problem. Different classes have their own responsibilities.

```
class InternetDownloader {
    public void downloadFromInternet() {
    }
}

class DataParser {
    public void parseData() {
    }
}

class DatabaseStorer {
    public void storeIntoDatabase() {
    }
}

class DownloadAndStore {
    void doEverything() {
        new InternetDownloader().downloadFromInternet();
        new DataParser().parseData();
        new DatabaseStorer().storeIntoDatabase();
    }
}
```

What is Encapsulation?

Encapsulation is “hiding the implementation of a Class behind a well defined interface”. Encapsulation helps us to change implementation of a class without breaking other code.

Approach 1

In this approach we create a public variable score. The main method directly accesses the score variable, updates it.

```
public class CricketScorer {
    public int score;
}

Let’s use the CricketScorer class.
public static void main(String[] args) {
      CricketScorer scorer = new CricketScorer();
      scorer.score = scorer.score + 4;
}
```

Approach 2

In this approach, we make score as private and access value through get and set methods. However, the logic of adding 4 to the score is performed in the main method.

```
public class CricketScorer {
    private int score;
    public int getScore() {
        return score;
    }

    public void setScore(int score) {
        this.score = score;
    }
}
```

Let’s use the CricketScorer class.

```
public static void main(String[] args) {
      CricketScorer scorer = new CricketScorer();

      int score = scorer.getScore();
      scorer.setScore(score + 4);
}
```

Approach 3

In this approach - For better encapsulation, the logic of doing the four operation also is moved to the CricketScorer class.

```
public class CricketScorer {
    private int score;

    public void four() {
        score += 4;
    }

}
```

Let’s use the CricketScorer class.

```
public static void main(String[] args) {
      CricketScorer scorer = new CricketScorer();
      scorer.four();
}
```

Description

In terms of encapsulation Approach 3 > Approach 2 > Approach 1. In Approach 3, the user of scorer class does not even know that there is a variable called score. Implementation of Scorer can change without changing other classes using Scorer.

What is Method Overloading?

A method having the same name as another method (in same class or a sub class) but having different parameters is called an Overloaded Method.

Example 1

doIt method is overloaded in the below example:

```
class Foo{
    public void doIt(int number){

    }
    public void doIt(String string){

    }
}
```

Example 2

Overloading can also be done from a sub class.

```
class Bar extends Foo{
    public void doIt(float number){
    }
}
```

What is Method Overriding?

Creating a Sub Class Method with same signature as that of a method in SuperClass is called Method Overriding.

Method Overriding Example 1:

Let’s define an Animal class with a method shout.

```
public class Animal {
    public String bark() {
        return "Don't Know!";
    }
}
```

Let’s create a sub class of Animal – Cat - overriding the existing shout method in Animal.

```
class Cat extends Animal {
    public String bark() {
        return "Meow Meow";
    }
}
```

bark method in Cat class is overriding the bark method in Animal class.

What is an Inner Class?

Inner Classes are classes which are declared inside other classes. Consider the following example:

```
class OuterClass {

    public class InnerClass {
    }

    public static class StaticNestedClass {
    }

}
```

What is a Static Inner Class?

A class declared directly inside another class and declared as static. In the example above, class name StaticNestedClass is a static inner class.

Can you create an inner class inside a method?

Yes. An inner class can be declared directly inside a method. In the example below, class name `MethodLocalInnerClass` is a method inner class.

```
class OuterClass {

    public void exampleMethod() {
        class MethodLocalInnerClass {
        };
    }

}
```

Constructors

Constructor (Youtube Video link - https://www.youtube.com/watch?v=XrdxGT2s9tc) is invoked whenever we create an instance(object) of a Class. We cannot create an object without a constructor. If we do not provide a constructor, compiler provides a default no-argument constructor.

What is a Default Constructor?

Default Constructor is the constructor that is provided by the compiler. It has no arguments. In the example below, there are no Constructors defined in the Animal class. Compiler provides us with a default constructor, which helps us create an instance of animal class.

```java
public class Animal {
    String name;

    public static void main(String[] args) {
        // Compiler provides this class with a default no-argument constructor.
        // This allows us to create an instance of Animal class.
        Animal animal = new Animal();
    }
}
```

How do you call a Super Class Constructor from a Constructor?

A constructor can call the constructor of a super class using the super() method call. Only constraint is that it should be the first statement.

Both example constructors below can replaces the no argument "public Animal() " constructor in Example 3.

```
public Animal() {
    super();
    this.name = "Default Name";
}
```

Can a constructor be called directly from a method?

A constructor cannot be explicitly called from any method except another constructor.

```
class Animal {
    String name;

    public Animal() {
    }

    public method() {
        Animal();// Compiler error
    }
}
```

Is a super class constructor called even when there is no explicit call from a sub class constructor?

If a super class constructor is not explicitly called from a sub class constructor, super class (no argument) constructor is automatically invoked (as first line) from a sub class constructor.

Consider the example below:

```java
class Animal {
    public Animal() {
        System.out.println("Animal Constructor");
    }
}

class Dog extends Animal {
    public Dog() {
        System.out.println("Dog Constructor");
    }
}

class Labrador extends Dog {
    public Labrador() {
        System.out.println("Labrador Constructor");
    }
}

public class ConstructorExamples {
    public static void main(String[] args) {
        Labrador labrador = new Labrador();
    }
}
```

Program Output

```
Animal Constructor
Dog Constructor
Labrador Constructor
```

Interface

What is an Interface?

An interface (YouTube video link - https://www.youtube.com/watch?v=VangB-sVNgg) defines a contract for responsibilities (methods) of a class.

How do you define an Interface?

An interface is declared by using the keyword interface. Look at the example below: Flyable is an interface.

```
//public abstract are not necessary
public abstract interface Flyable {
    //public abstract are not necessary
    public abstract void fly();
}
```

How do you implement an interface?

We can define a class implementing the interface by using the implements keyword. Let us look at a couple of examples:

Example 1

Class Aeroplane implements Flyable and implements the abstract method fly().

```
public class Aeroplane implements Flyable{
    @Override
    public void fly() {
        System.out.println("Aeroplane is flying");
    }
}
```

Example 2

```
public class Bird implements Flyable{
    @Override
    public void fly() {
        System.out.println("Bird is flying");
    }
}
```

Can you tell a little bit more about interfaces?

Variables in an interface are always public, static, final. Variables in an interface cannot be declared private.

```
interface ExampleInterface1 {
    //By default - public static final. No other modifier allowed
    //value1,value2,value3,value4 all are - public static final
    int value1 = 10;
    public int value2 = 15;
    public static int value3 = 20;
    public static final int value4 = 25;
    //private int value5 = 10;//COMPILER ERROR
}
```

Interface methods are by default public and abstract. A concrete method (fully defined method) cannot be created in an interface. Consider the example below:

```
interface ExampleInterface1 {
    //By default - public abstract. No other modifier allowed
    void method1();//method1 is public and abstract
    //private void method6();//COMPILER ERROR!

    /*//Interface cannot have body (definition) of a method
```

```
      //This method, uncommented, gives COMPILER ERROR!
    void method5() {
        System.out.println("Method5");
    }
     */
}
```

Can you extend an interface?

An interface can extend another interface. Consider the example below:

```
interface SubInterface1 extends ExampleInterface1{
    void method3();
}
```

Class implementing SubInterface1 should implement both methods - method3 and method1(from ExampleInterface1)

An interface cannot extend a class.

```
/* //COMPILE ERROR IF UnCommented
   //Interface cannot extend a Class
interface SubInterface2 extends Integer{
    void method3();
}
*/
```

Can a class extend multiple interfaces?

A class can implement multiple interfaces. It should implement all the method declared in all Interfaces being implemented.

```
interface ExampleInterface2 {
    void method2();
```

```java
}

class SampleImpl implements ExampleInterface1,ExampleInterface2{
    /* A class should implement all the methods in an interface.
       If either of method1 or method2 is commented, it would
       result in compilation error.
     */
    public void method2() {
        System.out.println("Sample Implementation for Method2");
    }

    public void method1() {
        System.out.println("Sample Implementation for Method1");
    }

}
```

Access Modifiers

What is default class modifier?

- A class is called a Default Class is when there is no access modifier specified on a class.
- Default classes are visible inside the same package only.
- Default access is also called Package access.

Example

```
package com.rithus.classmodifiers.defaultaccess.a;

/* No public before class. So this class has default access*/
class DefaultAccessClass {
//Default access is also called package access
}
```

Another Class in Same Package: Has access to default class

```
package com.rithus.classmodifiers.defaultaccess.a;

public class AnotherClassInSamePackage {
    //DefaultAccessClass and AnotherClassInSamePackage
    //are in same package.
    //So, DefaultAccessClass is visible.
    //An instance of the class can be created.
    DefaultAccessClass defaultAccess;
}
```

Class in Different Package: NO access to default class

```
package com.rithus.classmodifiers.defaultaccess.b;
```

```
public class ClassInDifferentPackage {
    //Class DefaultAccessClass and Class ClassInDifferentPackage
    //are in different packages (*.a and *.b)
    //So, DefaultAccessClass is not visible to ClassInDifferentPackage

    //Below line of code will cause compilation error if uncommented
    //DefaultAccessClass defaultAccess; //COMPILE ERROR!!
}
```

What are the different method access modifiers?

Let's discuss about access modifiers in order of increasing access.

private

a. Private variables and methods can be accessed only in the class they are declared.

b. Private variables and methods from SuperClass are NOT available in SubClass.

default or package

a. Default variables and methods can be accessed in the same package Classes.

b. Default variables and methods from SuperClass are available only to SubClasses in same package.

protected

a. Protected variables and methods can be accessed in the same package Classes.

b. Protected variables and methods from SuperClass are available to SubClass in any package

public

a. Public variables and methods can be accessed from every other Java classes.

b. Public variables and methods from SuperClass are all available directly in the SubClass

What is the use of a final modifier on a class?

Final class cannot be extended. Example of Final class in Java is the String class. Final is used very rarely as it prevents re-use of the class.Consider the class below which is declared as final.

```java
final public class FinalClass {
}
```

Below class will not compile if uncommented. FinalClass cannot be extended.

```java
/*
class ExtendingFinalClass extends FinalClass{ //COMPILER ERROR

}
*/
```

What is the use of a final modifier on a method?

Final methods cannot be overridden. Consider the class FinalMemberModifiersExample with method finalMethod which is declared as final.

```
public class FinalMemberModifiersExample {
    final void finalMethod(){
    }
}
```

Any SubClass extending above class cannot override the finalMethod().

```
class SubClass extends FinalMemberModifiersExample {
    //final method cannot be over-riddent
    //Below method, uncommented, causes compilation Error
    /*
    final void finalMethod(){

    }
    */
}
```

What is a Final variable?

Once initialized, the value of a final variable cannot be changed.

```
final int finalValue = 5;
//finalValue = 10; //COMPILER ERROR
```

What is a final argument?

Final arguments value cannot be modified. Consider the example below:

```
void testMethod(final int finalArgument){
    //final argument cannot be modified
    //Below line, uncommented, causes compilation Error
    //finalArgument = 5;//COMPILER ERROR
}
```

What happens when a variable is marked as volatile?

- Volatile can only be applied to instance variables.
- A volatile variable is one whose value is always written to and read from "main memory". Each thread has its own cache in Java. The volatile variable will not be stored on a Thread cache.

What is a Static Variable?

Static variables and methods are class level variables and methods. There is only one copy of the static variable for the entire Class. Each instance of the Class (object) will NOT have a unique copy of a static variable. Let's start with a real world example of a Class with static variable and methods.

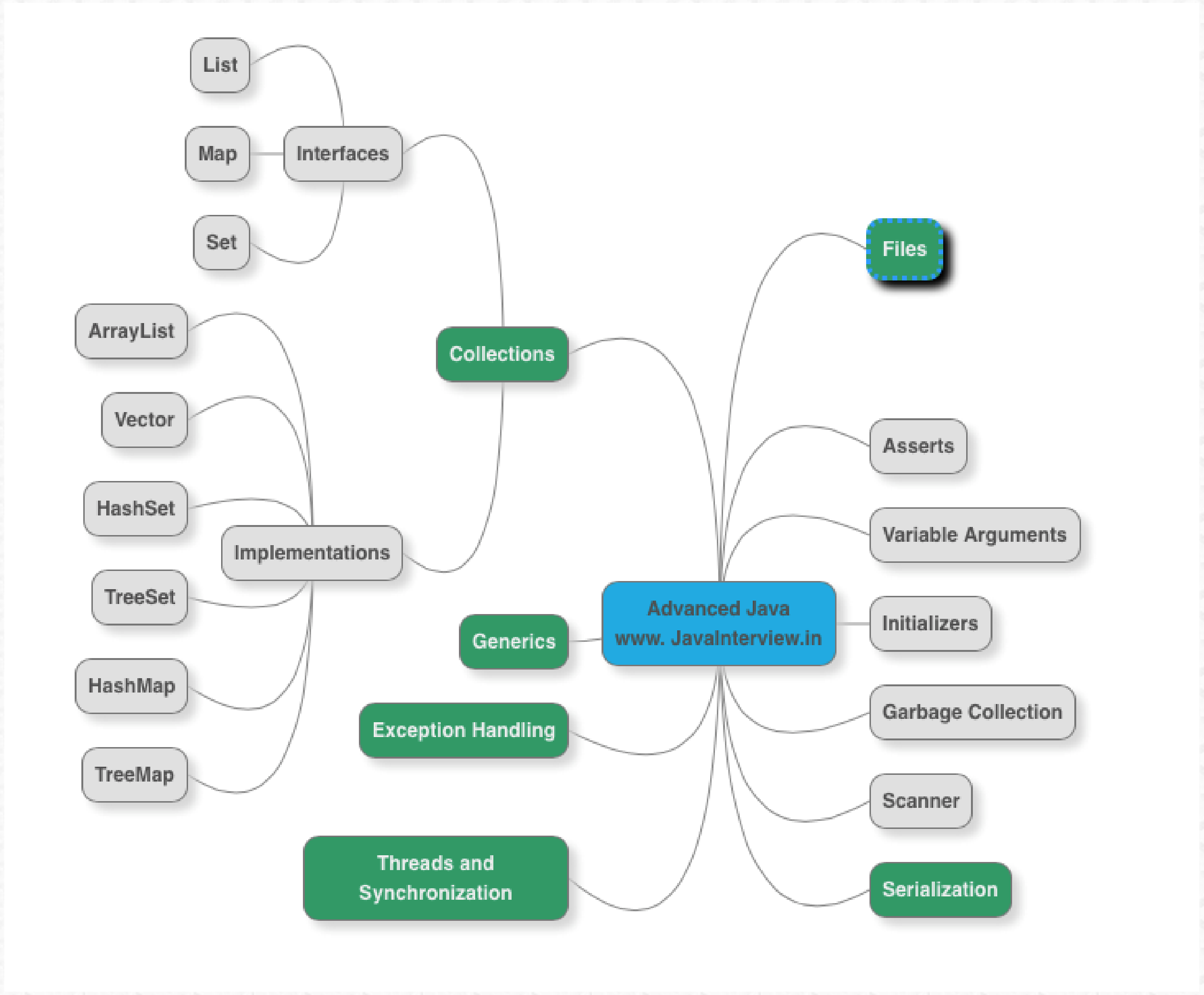
Static Variable/Method – Example

count variable in Cricketer class is static. The method to get the count value getCount() is also a static method.

```
public class Cricketer {
    private static int count;

    public Cricketer() {
        count++;
    }

    static int getCount() {
        return count;
    }

    public static void main(String[] args) {

        Cricketer cricketer1 = new Cricketer();
        Cricketer cricketer2 = new Cricketer();
        Cricketer cricketer3 = new Cricketer();
        Cricketer cricketer4 = new Cricketer();

        System.out.println(Cricketer.getCount());//4
    }
}
```

4 instances of the Cricketer class are created. Variable count is incremented with every instance created in the constructor.

Advanced Java



What are Variable Arguments or varargs?

Variable Arguments allow calling a method with different number of parameters. Consider the example method sum below. This sum method can be called with 1 int parameter or 2 int parameters or more int parameters.

```
//int(type) followed ... (three dot's) is syntax of a variable argument.
public int sum(int... numbers) {
    //inside the method a variable argument is similar to an array.
    //number can be treated as if it is declared as int[] numbers;
    int sum = 0;
    for (int number: numbers) {
        sum += number;
    }
    return sum;
}

public static void main(String[] args) {
    VariableArgumentExamples example = new VariableArgumentExamples();
    //3 Arguments
    System.out.println(example.sum(1, 4, 5));//10
    //4 Arguments
    System.out.println(example.sum(1, 4, 5, 20));//30
    //0 Arguments
    System.out.println(example.sum());//0
}
```

What are Asserts used for?

Assertions are introduced in Java 1.4. They enable you to validate assumptions. If an assert fails (i.e. returns false), AssertionError is thrown (if assertions are enabled). Basic assert is shown in the example below

```java
private int computerSimpleInterest(int principal,float interest,int years){
    assert(principal>0);
    return 100;
}
```

When should Asserts be used?

Assertions should not be used to validate input data to a public method or command line argument. IllegalArgumentException would be a better option. In public method, only use assertions to check for cases which are never supposed to happen.

What is Garbage Collection?

Garbage Collection is a name given to automatic memory management in Java. Aim of Garbage Collection is to Keep as much of heap available (free) for the program as possible. JVM removes objects on the heap which no longer have references from the heap.

Can you explain Garbage Collection with an example?

Let's say the below method is called from a function.

```java
void method(){
    Calendar calendar = new GregorianCalendar(2000,10,30);
    System.out.println(calendar);
}
```

An object of the class GregorianCalendar is created on the heap by the first line of the function with one reference variable calendar.

After the function ends execution, the reference variable calendar is no longer valid. Hence, there are no references to the object created in the method.

JVM recognizes this and removes the object from the heap. This is called Garbage Collection.

When is Garbage Collection run?

Garbage Collection runs at the whims and fancies of the JVM (it isn't as bad as that). Possible situations when Garbage Collection might run are

- when available memory on the heap is low
- when cpu is free

What are best practices on Garbage Collection?

Programmatically, we can request (remember it's just a request - Not an order) JVM to run Garbage Collection by calling System.gc() method.

JVM might throw an OutOfMemoryException when memory is full and no objects on the heap are eligible for garbage collection.

finalize() method on the objected is run before the object is removed from the heap from the garbage collector. We recommend not to write any code in finalize();

What are Initialization Blocks?

Initialization Blocks - Code which runs when an object is created or a class is loaded

There are two types of Initialization Blocks

- **Static Initializer**: Code that runs when a class is loaded.
- **Instance Initializer**: Code that runs when a new object is created.

What is a Static Initializer?

Look at the example below:

```
public class InitializerExamples {
static int count;
int i;

static{
    //This is a static initializers. Run only when Class is first loaded.
    //Only static variables can be accessed
    System.out.println("Static Initializer");
    //i = 6;//COMPILER ERROR
    System.out.println("Count when Static Initializer is run is " + count);
}

public static void main(String[] args) {
    InitializerExamples example = new InitializerExamples();
       InitializerExamples example2 = new InitializerExamples();
       InitializerExamples example3 = new InitializerExamples();
}
}
```

Code within static{ and } is called a static initializer. This is run only when class is first loaded. Only static variables can be accessed in a static initializer.

Example Output

```
Static Initializer
Count when Static Initializer is run is 0
```

Even though three instances are created static initializer is run only once.

What is an Instance Initializer Block?

Let's look at an example

```java
public class InitializerExamples {
    static int count;
    int i;
    {
        //This is an instance initializers. Run every time an object is created.
        //static and instance variables can be accessed
        System.out.println("Instance Initializer");
        i = 6;
        count = count + 1;
        System.out.println("Count when Instance Initializer is run is " + count);
    }

    public static void main(String[] args) {
        InitializerExamples example = new InitializerExamples();
        InitializerExamples example1 = new InitializerExamples();
        InitializerExamples example2 = new InitializerExamples();
    }

}
```

Code within instance initializer is run every time an instance of the class is created.

Example Output

```
Instance Initializer
Count when Instance Initializer is run is 1
Instance Initializer
Count when Instance Initializer is run is 2
Instance Initializer
Count when Instance Initializer is run is 3
```

What are Regular Expressions?

Regular Expressions make parsing, scanning and splitting a string very easy. We will first look at how you can evaluate a regular expressions in Java – using Patter, Matcher and Scanner classes. We will then look into how to write a regular expression.

What is Tokenizing?

Tokenizing means splitting a string into several sub strings based on delimiters. For example, delimiter ; splits the string ac;bd;def;e into four sub strings ac, bd, def and e.

Delimiter can in itself be any of the regular expression(s) we looked at earlier.

String.split(regex) function takes regex as an argument.

Can you give an example of Tokenizing?

```
private static void tokenize(String string,String regex) {
    String[] tokens = string.split(regex);
    System.out.println(Arrays.toString(tokens));
}
```

Example:

```
tokenize("ac;bd;def;e",";");//[ac, bd, def, e]
```

How can you Tokenize using Scanner Class?

```
private static void tokenizeUsingScanner(String string,String regex) {
    Scanner scanner = new Scanner(string);
    scanner.useDelimiter(regex);
    List<String> matches = new ArrayList<String>();
    while(scanner.hasNext()){
        matches.add(scanner.next());
    }
    System.out.println(matches);
}
```

Example:

```
tokenizeUsingScanner("ac;bd;def;e",";");//[ac, bd, def, e]
```

How do you add hours to a date object?

For more details about Date, refer to this youtube video. Lets now look at adding a few hours to a date object. All date manipulation to date needs to be done by adding milliseconds to the date. For example, if we want to add 6 hour, we convert 6 hours into millseconds. 6 hours = 6 * 60 * 60 * 1000 milliseconds. Below examples shows specific code.

```java
Date date = new Date();

//Increase time by 6 hrs
date.setTime(date.getTime() + 6 * 60 * 60 * 1000);
System.out.println(date);

//Decrease time by 6 hrs
date = new Date();
date.setTime(date.getTime() - 6 * 60 * 60 * 1000);
System.out.println(date);
```

How do you format Date Objects?

Formatting Dates is done by using DateFormat class. Let's look at a few examples.

```java
//Formatting Dates
System.out.println(DateFormat.getInstance().format(
        date));//10/16/12 5:18 AM
```

Formatting Dates with a locale

```java
System.out.println(DateFormat.getDateInstance(
        DateFormat.FULL, new Locale("it", "IT"))
        .format(date));//marted“ 16 ottobre 2012

System.out.println(DateFormat.getDateInstance(
        DateFormat.FULL, Locale.ITALIAN)
        .format(date));//marted“ 16 ottobre 2012

//This uses default locale US
System.out.println(DateFormat.getDateInstance(
        DateFormat.FULL).format(date));//Tuesday, October 16, 2012

System.out.println(DateFormat.getDateInstance()
        .format(date));//Oct 16, 2012
System.out.println(DateFormat.getDateInstance(
        DateFormat.SHORT).format(date));//10/16/12
System.out.println(DateFormat.getDateInstance(
        DateFormat.MEDIUM).format(date));//Oct 16, 2012

System.out.println(DateFormat.getDateInstance(
        DateFormat.LONG).format(date));//October 16, 2012
```

What is the use of Calendar class in Java?

Calendar class (Youtube video link - https://www.youtube.com/watch?v=hvnlYbt1ve0) is used in Java to manipulate Dates. Calendar class provides easy ways to add or reduce days, months or years from a date. It also provide lot of details about a date (which day of the year? Which week of the year? etc.)

How do you get an instance of Calendar class in Java?

Calendar class cannot be created by using new Calendar. The best way to get an instance of Calendar class is by using getInstance() static method in Calendar.

```
//Calendar calendar = new Calendar(); //COMPILER ERROR
Calendar calendar = Calendar.getInstance();
```

Can you explain some of the important methods in Calendar class?

Setting day, month or year on a calendar object is simple. Call the set method with appropriate Constant for Day, Month or Year. Next parameter is the value.

```
calendar.set(Calendar.DATE, 24);
calendar.set(Calendar.MONTH, 8);//8 - September
calendar.set(Calendar.YEAR, 2010);
```

Calendar get method

Let's get information about a particular date - 24th September 2010. We use the calendar get method. The parameter passed indicates what value we would want to get from the calendar – day or month or year or .. Few examples of the values you can obtain from a calendar are listed below.

```
System.out.println(calendar.get(Calendar.YEAR));//2010
System.out.println(calendar.get(Calendar.MONTH));//8
System.out.println(calendar.get(Calendar.DATE));//24
System.out.println(calendar.get(Calendar.WEEK_OF_MONTH));//4
System.out.println(calendar.get(Calendar.WEEK_OF_YEAR));//39
System.out.println(calendar.get(Calendar.DAY_OF_YEAR));//267
System.out.println(calendar.getFirstDayOfWeek());//1 -> Calendar.SUNDAY
```

What is the use of NumberFormat class?

Number format is used to format a number to different locales and different formats.

Format number Using Default locale

```
System.out.println(NumberFormat.getInstance().format(321.24f));//321.24
```

Format number using locale

Formatting a number using Netherlands locale

```
System.out.println(NumberFormat.getInstance(new Locale("nl")).format(4032.3f));//4.032,3
```

Formatting a number using Germany locale

```
System.out.println(NumberFormat.getInstance(Locale.GERMANY).format(4032.3f));//4.032,3
```

Formatting a Currency using Default locale

```
System.out.println(NumberFormat.getCurrencyInstance().format(40324.31f));//$40,324.31
```

Format currency using locale

Formatting a Currency using Netherlands locale

```
System.out.println(NumberFormat.getCurrencyInstance(new Locale("nl")).format(40324.31f));//? 40.324,31
```

Collections Interfaces

We will discuss about different collection interfaces along with their purpose. Refer to this youtube videos (https://www.youtube.com/watch?v=GnR4hCvEIJQ & https://www.youtube.com/watch?v=6dKGpOKAQqs) for more details.

Why do we need Collections in Java?

Arrays are not dynamic. Once an array of a particular size is declared, the size cannot be modified. To add a new element to the array, a new array has to be created with bigger size and all the elements from the old array copied to new array.

Collections are used in situations where data is dynamic. Collections allow adding an element, deleting an element and host of other operations. There are a number of Collections in Java allowing to choose the right Collection for the right context.

What are the important methods that are declared in the Collection Interface?

Most important methods declared in the collection interface are the methods to add and remove an element. add method allows adding an element to a collection and delete method allows deleting an element from a collection.

size() methods returns number of elements in the collection. Other important methods defined as part of collection interface are shown below.

```
interface Collection<E> extends Iterable<E>
{
  boolean add(E paramE);
  boolean remove(Object paramObject);

  int size();
  boolean isEmpty();
  void clear();

  boolean contains(Object paramObject);
  boolean containsAll(Collection<?> paramCollection);

  boolean addAll(Collection<? extends E> paramCollection);
  boolean removeAll(Collection<?> paramCollection);
  boolean retainAll(Collection<?> paramCollection);


  Iterator<E> iterator();

  //A NUMBER OF OTHER METHODS AS WELL..
}
```

Can you explain briefly about the List Interface?

List interface extends Collection interface. So, it contains all methods defined in the Collection interface. In addition, List interface allows operation specifying the position of the element in the Collection.

Most important thing to remember about a List interface - any implementation of the List interface would maintain the insertion order. When an element A is inserted into

a List (without specifying position) and then another element B is inserted, A is stored before B in the List.
When a new element is inserted without specifying a position, it is inserted at the end of the list of elements.

However, We can also use the void add(int position, E paramE); method to insert an element at a specific position.
Listed below are some of the important methods in the List interface (other than those inherited from Collection interface):

```
interface List<E> extends Collection<E>
{
  boolean addAll(int paramInt, Collection<? extends E> paramCollection);

  E get(int paramInt);
  E set(int paramInt, E paramE);

  void add(int paramInt, E paramE);
  E remove(int paramInt);

  int indexOf(Object paramObject);
  int lastIndexOf(Object paramObject);

  ListIterator<E> listIterator();
  ListIterator<E> listIterator(int paramInt);
  List<E> subList(int paramInt1, int paramInt2);
}
```

Can you briefly explain about the Map Interface?

First and foremost, Map interface does not extend Collection interface. So, it does not inherit any of the methods from the Collection interface.

A Map interface supports Collections that use a key value pair. A key-value pair is a set of linked data items: a key, which is a unique identifier for some item of data, and the value, which is either the data or a pointer to the data. Key-value pairs are used in lookup tables, hash tables and configuration files. A key value pair in a Map interface is called an Entry.

Put method allows to add a key, value pair to the Map.

```
  V put(K paramK, V paramV);
```

Get method allows to get a value from the Map based on the key.

```
  V get(Object paramObject);
```

Other important methods in Map Inteface are shown below:

```
interface Map<K, V>
{
  int size();
  boolean isEmpty();

  boolean containsKey(Object paramObject);
  boolean containsValue(Object paramObject);

  V get(Object paramObject);
  V put(K paramK, V paramV);
  V remove(Object paramObject);

  void putAll(Map<? extends K, ? extends V> paramMap);
  void clear();

  Set<K> keySet();
  Collection<V> values();
  Set<Entry<K, V>> entrySet();

  boolean equals(Object paramObject);
  int hashCode();

  public static abstract interface Entry<K, V>
  {
    K getKey();
    V getValue();
    V setValue(V paramV);
    boolean equals(Object paramObject);
    int hashCode();
  }
}
```

What is the difference between Set and SortedSet?

SortedSet Interface extends the Set Interface. Both Set and SortedSet do not allow duplicate elements.

Main difference between Set and SortedSet is - an implementation of SortedSet interface maintains its elements in a sorted order. Set interface does not guarantee any Order. For example, If elements 4,5,3 are inserted into an implementation of Set interface, it might store the elements in any order. However, if we use SortedSet, the elements are sorted. The SortedSet implementation would give an output 3,4,5.

Important Operations in the SortedSet interface which are not present in the Set Interface are listed below:

```
public interface SortedSet<E> extends Set<E> {

    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    E first();
    E last();

    Comparator<? super E> comparator();
}
```

What is difference between Map and SortedMap?

SortedMap interface extends the Map interface. In addition, an implementation of SortedMap interface maintains keys in a sorted order.

Methods are available in the interface to get a ranges of values based on their keys.

```
public interface SortedMap<K, V> extends Map<K, V> {
    Comparator<? super K> comparator();

    SortedMap<K, V> subMap(K fromKey, K toKey);

    SortedMap<K, V> headMap(K toKey);

    SortedMap<K, V> tailMap(K fromKey);

    K firstKey();

    K lastKey();
}
```

Explain briefly about Queue Interface?

Queue Interface extends Collection interface. Queue Interface is typically used for implementation holding elements in order for some processing.

Queue interface offers methods peek() and poll() which get the element at head of the queue. The difference is that poll() method removes the head from queue also. peek() would keep head of the queue unchanged.

```
interface Queue<E> extends Collection<E>
{
  boolean offer(E paramE);
  E remove();
  E poll();
  E element();
  E peek();
}
```

Explain briefly about Iterator?

Iterator interface enables us to iterate (loop around) a collection. All collections define a method iterator() that gets an iterator of the collection.

hasNext() checks if there is another element in the collection being iterated. next() gets the next element.

```
public interface Iterator<E> {
    boolean hasNext();

    E next();
}
```

Explain about ArrayList with an example?

Refer to this video - https://www.youtube.com/watch?v=_JTIYhnLemA for more details about ArrayList. Let us look at a few important interview questions.

ArrayList implements the list interface. So, ArrayList stores the elements in insertion order (by default). Element's can be inserted into and removed from ArrayList based on their position.

Let's look at how to instantiate an ArrayList of integers.

```
List<Integer> integers = new ArrayList<Integer>();
```

Code like below is permitted because of auto boxing. 5 is auto boxed into Integer object and stored in ArrayList.

```
integers.add(5);//new Integer(5)
```

Add method (by default) adds the element at the end of the list.

Can an ArrayList have Duplicate elements?

ArrayList can have duplicates (since List can have duplicates).

```
List<String> arraylist = new ArrayList<String>();

//adds at the end of list
arraylist.add("Sachin");//[Sachin]

//adds at the end of list
```

```
arraylist.add("Dravid");//[Sachin, Dravid]

//adds at the index 0
arraylist.add(0, "Ganguly");//[Ganguly, Sachin, Dravid]

//List allows duplicates - Sachin is present in the list twice
arraylist.add("Sachin");//[ Ganguly, Sachin, Dravid, Sachin]

System.out.println(arraylist.size());//4
System.out.println(arraylist.contains("Dravid"));//true
```

How do you iterate around an ArrayList using Iterator?

Example below shows how to iterate around an ArrayList.

```
Iterator<String> arraylistIterator = arraylist
        .iterator();
while (arraylistIterator.hasNext()) {
    String str = arraylistIterator.next();
    System.out.println(str);//Prints the 4 names in the list on separate lines.
}
```

How do you sort an ArrayList?

Example below shows how to sort an ArrayList. It uses the Collections.sort method.

```
List<String> numbers = new ArrayList<String>();
numbers.add("one");
numbers.add("two");
numbers.add("three");
numbers.add("four");
System.out.println(numbers);//[one, two, three, four]

//Strings - By Default - are sorted alphabetically
Collections.sort(numbers);

System.out.println(numbers);//[four, one, three, two]
```

How do you sort elements in an ArrayList using Comparable interface?

Consider the following class Cricketer.

```
class Cricketer implements Comparable<Cricketer> {
    int runs;
    String name;

    public Cricketer(String name, int runs) {
        super();
        this.name = name;
        this.runs = runs;
    }

    @Override
    public String toString() {
        return name + " " + runs;
    }

    @Override
    public int compareTo(Cricketer that) {
        if (this.runs > that.runs) {
            return 1;
        }
        if (this.runs < that.runs) {
            return -1;
        }
        return 0;
    }
}
```

Let's now try to sort a list containing objects of Cricketer class.

```
List<Cricketer> cricketers = new ArrayList<Cricketer>();
cricketers.add(new Cricketer("Bradman", 9996));
cricketers.add(new Cricketer("Sachin", 14000));
cricketers.add(new Cricketer("Dravid", 12000));
cricketers.add(new Cricketer("Ponting", 11000));
System.out.println(cricketers);
//[Bradman 9996, Sachin 14000, Dravid 12000, Ponting 11000]
```

Now let's try to sort the cricketers.

```
Collections.sort(cricketers);
System.out.println(cricketers);
//[Bradman 9996, Ponting 11000, Dravid 12000, Sachin 14000]
```

How do you sort elements in an ArrayList using Comparator interface?

Other option to sort collections is by creating a separate class which implements Comparator interface.
Example below:

```
class DescendingSorter implements Comparator<Cricketer> {

    //compareTo returns -1 if cricketer1 < cricketer2
    //                   1 if cricketer1 > cricketer2
    //                   0 if cricketer1 = cricketer2

    //Since we want to sort in descending order,
    //we should return -1 when runs are more
    @Override
    public int compare(Cricketer cricketer1,
            Cricketer cricketer2) {
        if (cricketer1.runs > cricketer2.runs) {
            return -1;
        }
        if (cricketer1.runs < cricketer2.runs) {
            return 1;
        }
        return 0;
    }

}
```

Let’s now try to sort the previous defined collection:

```
Collections
        .sort(cricketers, new DescendingSorter());

System.out.println(cricketers);
//[Sachin 14000, Dravid 12000, Ponting 11000, Bradman 9996]
```

How do you convert List to Array?

There are two ways. First is to use toArray(String) function. Example below. This creates an array of String's

```
List<String> numbers1 = new ArrayList<String>();
numbers1.add("one");
numbers1.add("two");
numbers1.add("three");
numbers1.add("four");
String[] numbers1Array = new String[numbers1.size()];
numbers1Array = numbers1.toArray(numbers1Array);
System.out.println(Arrays.toString(numbers1Array));
//prints [one, two, three, four]
```

Other is to use toArray() function. Example below. This creates an array of Objects.

```
Object[] numbers1ObjArray = numbers1.toArray();
System.out.println(Arrays
        .toString(numbers1ObjArray));
//[one, two, three, four]
```

How do you convert an Array to List?

```
String values[] = { "value1", "value2", "value3" };
List<String> valuesList = Arrays.asList(values);
System.out.println(valuesList);//[value1, value2, value3]
```

Following set of videos deal with collections interview questions in great detail : Video1, Video2 & Video3

What is Vector class? How is it different from an ArrayList?

Vector has the same operations as an ArrayList. However, all methods in Vector are synchronized. So, we can use Vector if we share a list between two threads and we would want to them synchronized.

What is LinkedList? What interfaces does it implement? How is it different from an ArrayList?

Linked List extends List and Queue.Other than operations exposed by the Queue interface, LinkedList has the same operations as an ArrayList. However, the underlying implementation of Linked List is different from that of an ArrayList.

ArrayList uses an Array kind of structure to store elements. So, inserting and deleting from an ArrayList are expensive operations. However, search of an ArrayList is faster than LinkedList.

LinkedList uses a linked representation. Each object holds a link to the next element. Hence, insertion and deletion are faster than ArrayList. But searching is slower.

Can you give examples of classes that implement the Set Interface?

HashSet, LinkedHashSet and TreeSet implement the Set interface. These classes are described in great detail in the video - https://www.youtube.com/watch?v=W5c8uXi4qTw.

What is a HashSet?

HashSet implements set interface. So, HashSet does not allow duplicates. However, HashSet does not support ordering. The order in which elements are inserted is not maintained.

HashSet Example

```
Set<String> hashset = new HashSet<String>();

hashset.add("Sachin");
System.out.println(hashset);//[Sachin]

hashset.add("Dravid");
System.out.println(hashset);//[Sachin, Dravid]
```

Let's try to add Sachin to the Set now. Sachin is Duplicate. So will not be added. returns false.

```
hashset.add("Sachin");//returns false since element is not added
System.out.println(hashset);//[Sachin, Dravid]
```

What is a LinkedHashSet? How is different from a HashSet?

LinkedHashSet implements set interface and exposes similar operations to a HashSet. Difference is that LinkedHashSet maintains insertion order. When we iterate a LinkedHashSet, we would get the elements back in the order in which they were inserted.

What is a TreeSet? How is different from a HashSet?

TreeSet implements Set, SortedSet and NavigableSet interfaces.TreeSet is similar to HashSet except that it stores element's in Sorted Order.

```
Set<String> treeSet = new TreeSet<String>();

treeSet.add("Sachin");
System.out.println(treeSet);//[Sachin]
```

Notice that the list is sorted after inserting Dravid.

```
//Alphabetical order
treeSet.add("Dravid");
System.out.println(treeSet);//[Dravid, Sachin]
```

Notice that the list is sorted after inserting Ganguly.

```
treeSet.add("Ganguly");
System.out.println(treeSet);//[Dravid, Ganguly, Sachin]

//Sachin is Duplicate. So will not be added. returns false.
treeSet.add("Sachin");//returns false since element is not added
System.out.println(treeSet);//[Dravid, Ganguly, Sachin]
```

Can you give examples of implementations of NavigableSet?

TreeSet implements this interface. Let's look at an example with TreeSet. Note that elements in TreeSet are sorted.

```
TreeSet<Integer> numbersTreeSet = new TreeSet<Integer>();
numbersTreeSet.add(55);
numbersTreeSet.add(25);
numbersTreeSet.add(35);
numbersTreeSet.add(5);
numbersTreeSet.add(45);
```

NavigableSet interface has following methods.

Lower method finds the highest element lower than specified element. Floor method finds the highest element lower than or equal to specified element. Corresponding methods for finding lowest number higher than specified element are higher and ceiling. A few examples using the Set created earlier below.

```
//Find the highest number which is lower than 25
System.out.println(numbersTreeSet.lower(25));//5

//Find the highest number which is lower than or equal to 25
System.out.println(numbersTreeSet.floor(25));//25

//Find the lowest number higher than 25
System.out.println(numbersTreeSet.higher(25));//35

//Find the lowest number higher than or equal to 25
System.out.println(numbersTreeSet.ceiling(25));//25
```

What are the different implementations of a Map Interface?

HashMap and TreeMap. These classes are explained in detail in this video - https://www.youtube.com/watch?v=ZNhT_Z8_q9s.

What is a HashMap?

HashMap implements Map interface – there by supporting key value pairs. Let's look at an example.

HashMap Example

```
Map<String, Cricketer> hashmap = new HashMap<String, Cricketer>();
hashmap.put("sachin",
        new Cricketer("Sachin", 14000));
hashmap.put("dravid",
        new Cricketer("Dravid", 12000));
hashmap.put("ponting", new Cricketer("Ponting",
        11500));
hashmap.put("bradman", new Cricketer("Bradman",
        9996));
```

What are the different methods in a Hash Map?

get method gets the value of the matching key.

```
System.out.println(hashmap.get("ponting"));//Ponting 11500

//if key is not found, returns null.
System.out.println(hashmap.get("lara"));//null
```

If existing key is reused, it would replace existing value with the new value passed in.

```
//In the example below, an entry with key "ponting" is already present.
//Runs are updated to 11800.
hashmap.put("ponting", new Cricketer("Ponting",
        11800));

//gets the recently updated value
System.out.println(hashmap.get("ponting"));//Ponting 11800
```

What is a TreeMap? How is different from a HashMap?

TreeMap is similar to HashMap except that it stores keys in sorted order. It implements NavigableMap interface and SortedMap interfaces along with the Map interface.

```
Map<String, Cricketer> treemap = new TreeMap<String, Cricketer>();
treemap.put("sachin",
        new Cricketer("Sachin", 14000));
System.out.println(treemap);
//{sachin=Sachin 14000}
```

We will now insert a Cricketer with key dravid. In sorted order,dravid comes before sachin. So, the value with key dravid is inserted at the start of the Map.

```
treemap.put("dravid",
        new Cricketer("Dravid", 12000));
System.out.println(treemap);
//{dravid=Dravid 12000, sachin=Sachin 14000}
```

We will now insert a Cricketer with key ponting. In sorted order, ponting fits in between dravid and sachin.

```
treemap.put("ponting", new Cricketer("Ponting",
        11500));
System.out.println(treemap);
```

```
//{dravid=Dravid 12000, ponting=Ponting 11500, sachin=Sachin 14000}

treemap.put("bradman", new Cricketer("Bradman",
        9996));
System.out.println(treemap);
//{bradman=Bradman 9996, dravid=Dravid 12000, ponting=Ponting 11500, sachin=Sachin 14000}
```

Can you give an example of implementation of NavigableMap Interface?

TreeMap is a good example of a NavigableMap interface implementation. Note that keys in TreeMap are sorted.

```
TreeMap<Integer, Cricketer> numbersTreeMap = new TreeMap<Integer, Cricketer>();
numbersTreeMap.put(55, new Cricketer("Sachin",
        14000));
numbersTreeMap.put(25, new Cricketer("Dravid",
        12000));
numbersTreeMap.put(35, new Cricketer("Ponting",
        12000));
numbersTreeMap.put(5,
        new Cricketer("Bradman", 9996));
numbersTreeMap
        .put(45, new Cricketer("Lara", 10000));
```

lowerKey method finds the highest key lower than specified key. floorKey method finds the highest key lower than or equal to specified key. Corresponding methods for finding lowest key higher than specified key are higher and ceiling. A few examples using the Map created earlier below.

```
//Find the highest key which is lower than 25
System.out.println(numbersTreeMap.lowerKey(25));//5

//Find the highest key which is lower than or equal to 25
```

```
System.out.println(numbersTreeMap.floorKey(25));//25

//Find the lowest key higher than 25
System.out.println(numbersTreeMap.higherKey(25));//35

//Find the lowest key higher than or equal to 25
System.out.println(numbersTreeMap.ceilingKey(25));//25
```

What is a PriorityQueue?

PriorityQueue implements the Queue interface.

```
//Using default constructor - uses natural ordering of numbers
//Smaller numbers have higher priority
PriorityQueue<Integer> priorityQueue = new PriorityQueue<Integer>();
```

Adding an element into priority queue - offer method

```
priorityQueue.offer(24);
priorityQueue.offer(15);
priorityQueue.offer(9);
priorityQueue.offer(45);

System.out.println(priorityQueue);//[9, 24, 15, 45]
```

Peek method examples

```
//peek method get the element with highest priority.
System.out.println(priorityQueue.peek());//9
//peek method does not change the queue
System.out.println(priorityQueue);//[9, 24, 15, 45]

//poll method gets the element with highest priority.
System.out.println(priorityQueue.poll());//9
//peek method removes the highest priority element from the queue.
System.out.println(priorityQueue);//[24, 15, 45]

//This comparator gives high priority to the biggest number.
```

```
Comparator reverseComparator = new Comparator<Integer>() {
    public int compare(Integer paramT1,
            Integer paramT2) {
        return paramT2 - paramT1;
    }

};
```

What are the static methods present in the Collections class?

- static int binarySearch(List, key)
  - Can be used only on sorted list
- static int binarySearch(List, key, Comparator)
- static void reverse(List)
  - Reverse the order of elements in a List.
- static Comparator reverseOrder();
  - Return a Comparator that sorts the reverse of the collection current sort sequence.
- static void sort(List)
- static void sort(List, Comparator)

Advanced Collections

What is the difference between synchronized and concurrent collections in Java?

Synchronized collections are implemented using synchronized methods and synchronized blocks. Only one thread can executing any of the synchronized code at a given point in time. This places severe restrictions on the concurrency of threads – thereby affecting performance of the application. All the pre Java 5 synchronized collections (HashTable & Vector, for example) use this approach.

Post Java 5, collections using new approaches to synchronization are available in Java. These are called concurrent collections. More details below.

Explain about the new concurrent collections in Java?

Post Java 5, collections using new approaches to synchronization are available in Java. These are called concurrent collections. Examples of new approaches are :

- Copy on Write
- Compare and Swap
- Locks

These new approaches to concurrency provide better performance in specific context's. We would discuss each of these approaches in detail below.

Explain about CopyOnWrite concurrent collections approach?

Important points about Copy on Write approach

- All values in collection are stored in an internal immutable (not-changeable) array. A new array is created if there is any modification to the collection.
- Read operations are not synchronized. Only write operations are synchronized.

Copy on Write approach is used in scenarios where reads greatly out number write's on a collection. CopyOnWriteArrayList & CopyOnWriteArraySet are implementations of this approach. Copy on Write collections are typically used in Subject – Observer scenarios, where the observers very rarely change. Most frequent operations would be iterating around the observers and notifying them.

What is CompareAndSwap approach?

Compare and Swap is one of the new approaches (Java 5) introduced in java to handle synchronization. In traditional approach, a method which modifies a member variable used by multiple threads is completely synchronized – to prevent other threads accessing stale value.

In compare and swap approach, instead of synchronizing entire method, the value of the member variable before calculation is cached. After the calculation, the cached value is compared with the current value of member variable. If the value is not modified, the calculated result is stored into the member variable. If another thread has modified the value, then the calculation can be performed again. Or skipped – as the need might be.

ConcurrentLinkedQueue uses this approach.

What is a Lock? How is it different from using synchronized approach?

When 10 methods are declared as synchronized, only one of them is executed by any of the threads at any point in time. This has severe performance impact.

Another new approach introduced in Java 5 is to use lock and unlock methods. Lock and unlock methods are used to divide methods into different blocks and help enhance concurrency. The 10 methods can be divided into different blocks, which can be synchronized based on different variables.

What is initial capacity of a Java Collection?

Extract from the reference : http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html. An instance of HashMap has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

As a general rule, the default load factor (.75) offers a good tradeoff between time and space costs. Higher values decrease the space overhead but increase the lookup cost (reflected in most of the operations of the HashMap class, including get and put).

The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the number of rehash operations.

What is load factor?

Refer answer to Initial Capacity above.

When does a Java collection throw UnsupportedOperationException?

All Java Collections extend Collection interface. So, they have to implement all the methods in the Collection interface. However, certain Java collections are optimized to be used in specific conditions and do not support all the Collection operations (methods). When an unsupported operation is called on a Collection, the Collection Implementation would throw an UnsupportedOperationException.

Arrays.asList returns a fixed-size list backed by the specified array. When an attempt is made to add or remove from this collection an UnsupportedOperationException is thrown. Below code throws UnsupportedOperationException.

```
List<String> list=Arrays.asList(new String[]{"ac","bddefe"});
```

```
list.remove();//throws UnsupportedOperationException
```

What is difference between fail-safe and fail-fast iterators?

Fail Fast Iterators throw a ConcurrentModificationException if there is a modification to the underlying collection is modified. This was the default behavior of the synchronized collections of pre Java 5 age.

Fail Safe Iterators do not throw exceptions even when there are changes in the collection. This is the default behavior of the concurrent collections, introduced since Java 5.

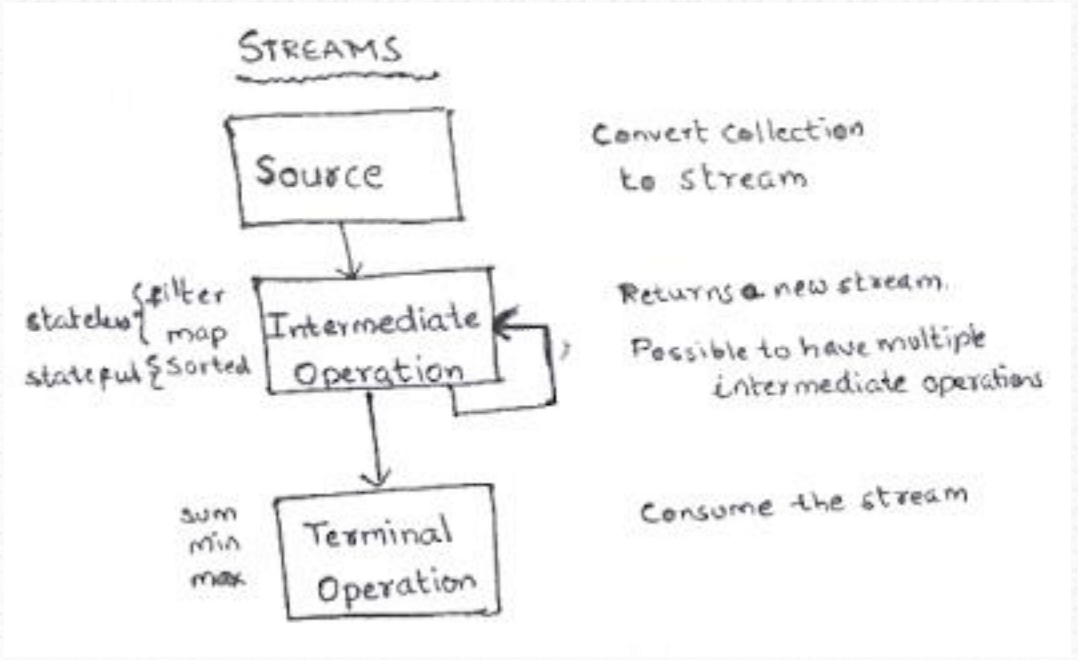
Explain about streams with an example?

Streams are introduced in Java 8. In combination with Lambda expressions, they attempt to bring some of the important functional programming concepts to Java.

A stream is a sequence of elements supporting sequential and parallel aggregate operations. Consider the example code below. Following steps are done:

- Step I : Creating an array as a stream
- Step II : Use Lambda Expression to create a filter
- Step III : Use map function to invoke a String function
- Step IV : Use sorted function to sort the array
- Step V : Print the array using forEach

```
Arrays.stream(new String[] {
     "Ram", "Robert", "Rahim"
})
     .filter(s - > s.startsWith("Ro"))
     .map(String::toLowerCase)
     .sorted()
     .forEach(System.out::println);
```



In general any use of streams involves

- Source - Creation or use of existing stream : Step I above
- Intermediate Operations - Step II, III and IV above. Intermediate Operations return a new stream
- Terminal Operation – Step V. Consume the stream. Print it to output or produce a result (sum,min,max etc).

Intermediate Operations are of two kinds

- Stateful : Elements need to be compared against one another (sort, distinct etc)
- Stateless : No need for comparing with other elements (map, filter etc)

What are atomic operations in Java?

Atomic Access Java Tutorial states “In programming, an atomic action is one that effectively happens all at once. An atomic action cannot stop in the middle: it either happens completely, or it doesn't happen at all. No side effects of an atomic action are visible until the action is complete”.

Let’s assume we are writing a multi threaded program. Let’s create an int variable i. Even a small operation, like i++ (increment), is not thread safe. i++ operation involves three steps.

1. Read the value which is currently stored in i
2. Add one to it (atomic operation).
3. Store it in i

In a multi-threaded environment, there can be unexpected results. For example, if thread1 is reading the value (step 1) and immediately after thread2 stores the value (step 3).

To prevent these, Java provides atomic operations. Atomic operations are performed as a single unit without interference from other threads ensuring data consistency.

A good example is AtomicInteger. To increment a value of AtomicInteger, we use the incrementAndGet() method. Java ensures this operation is Atomic.

What is BlockedQueue in Java?

BlockedQueue interface is introduced in Java specifically to address specific needs of some Producer Consumer scenarios. BlockedQueue allows the consumer to wait (for a specified time or infinitely) for an element to become available.

Generics

What are Generics?

Generics are used to create Generic Classes and Generic methods which can work with different Types(Classes).

Why do we need Generics? Can you give an example of how Generics make a program more flexible?

Consider the class below:

```java
class MyList {
    private List<String> values;

    void add(String value) {
        values.add(value);
    }

    void remove(String value) {
        values.remove(value);
    }
}
```

MyList can be used to store a list of Strings only.

```java
        MyList myList = new MyList();
        myList.add("Value 1");
        myList.add("Value 2");
```

To store integers, we need to create a new class. This is problem that Generics solve. Instead of hard-coding String class as the only type the class can work with, we make the class type a parameter to the class.

Example with Generics

Let's replace String with T and create a new class. Now the MyListGeneric class can be used to create a list of Integers or a list of Strings

```
class MyListGeneric<T> {
    private List<T> values;

    void add(T value) {
        values.add(value);
    }

    void remove(T value) {
        values.remove(value);
    }

    T get(int index) {
        return values.get(index);
    }
}

MyListGeneric<String> myListString = new MyListGeneric<String>();
myListString.add("Value 1");
myListString.add("Value 2");

MyListGeneric<Integer> myListInteger = new MyListGeneric<Integer>();
myListInteger.add(1);
myListInteger.add(2);
```

How do you declare a Generic Class?

Note the declaration of class:

```
class MyListGeneric<T>
```

Instead of T, We can use any valid identifier

What are the restrictions in using generic type that is declared in a class declaration?

If a generic is declared as part of class declaration, it can be used any where a type can be used in a class - method (return type or argument), member variable etc. For Example: See how T is used as a parameter and return type in the class MyListGeneric.

How can we restrict Generics to a subclass of particular class?

In MyListGeneric, Type T is defined as part of class declaration. Any Java Type can be used a type for this class. If we would want to restrict the types allowed for a Generic Type, we can use a Generic Restrictions. Consider the example class below: In declaration of the class, we specified a constraint "T extends Number". We can use the class MyListRestricted with any class extending (any sub class of) Number - Float, Integer, Double etc.

```
class MyListRestricted<T extends Number> {
    private List<T> values;

    void add(T value) {
        values.add(value);
    }

    void remove(T value) {
        values.remove(value);
    }

    T get(int index) {
        return values.get(index);
    }
}

MyListRestricted<Integer> restrictedListInteger = new MyListRestricted<Integer>();

restrictedListInteger.add(1);
restrictedListInteger.add(2);
```

String not valid substitute for constraint "T extends Number".

```
//MyListRestricted<String> restrictedStringList =
//                  new MyListRestricted<String>();//COMPILER ERROR
```

How can we restrict Generics to a super class of particular class?

In MyListGeneric, Type T is defined as part of class declaration. Any Java Type can be used a type for this class. If we would want to restrict the types allowed for a Generic Type, we can use a Generic Restrictions. In declaration of the class, we specified a constraint "T super Number". We can use the class MyListRestricted with any class that is a super class of Number class.

Can you give an example of a Generic Method?

A generic type can be declared as part of method declaration as well. Then the generic type can be used anywhere in the method (return type, parameter type, local or block variable type).

Consider the method below:

```java
    static <X extends Number> X doSomething(X number){
        X result = number;
        //do something with result
        return result;
    }
```

The method can now be called with any Class type extend Number.

```java
Integer i = 5;
Integer k = doSomething(i);
```

Exception Handling

Explain about Exception Handling with an example.

Exception Handling helps us to recover from an unexpected situations – File not found or network connection is down. The important part in exception handling is the try – catch block. Look at the example below.

```java
public static void main(String[] args) {
    method1();
    System.out.println("Line after Exception - Main");
}

private static void method1() {
    method2();
    System.out.println("Line after Exception - Method 1");
}

private static void method2() {
    try {
        String str = null;
        str.toString();
        System.out.println("Line after Exception - Method 2");
    } catch (Exception e) {
        // NOT PRINTING EXCEPTION TRACE- BAD PRACTICE
        System.out.println("Exception Handled - Method 2");
    }
}
```

Program Output

```
Exception Handled - Method 2
```

```
Line after Exception - Method 1
Line after Exception - Main
```

When exception is handled in a method, the calling methods will not need worry about that exception. Since Exception Handling is added in the method method2, the exception did not propogate to method1 i.e. method1 does not know about the exception in method2.

Few important things to remember from this example.

- If exception is handled, it does not propogate further.
- In a try block, the lines after the line throwing the exception are not executed.

What is the use of finally block in Exception Handling?

When an exception happens, the code after the line throwing exception is not executed. If code for things like closing a connection is present in these lines of code, it is not executed. This leads to connection and other resource leaks.

Code written in finally block is executed even when there is an exception.

Consider the example below. This is code without a finally block . We have Connection class with open and close methods. An exception happens in the main method. The connection is not closed because there is no finally block.

```java
class Connection {
    void open() {
        System.out.println("Connection Opened");
    }

    void close() {
        System.out.println("Connection Closed");
    }
}

public class ExceptionHandlingExample1 {

    public static void main(String[] args) {
        try {
            Connection connection = new Connection();
            connection.open();

            // LOGIC
            String str = null;
            str.toString();

            connection.close();
        } catch (Exception e) {
            // NOT PRINTING EXCEPTION TRACE- BAD PRACTICE
            System.out.println("Exception Handled - Main");
        }
    }
}
```

Output

```
Connection Opened
Exception Handled - Main
```

Connection that is opened is not closed. This results in a dangling (un-closed) connection.

Finally block is used when code needs to be executed irrespective of whether an exception is thrown. Let us now move connection.close(); into a finally block. Also connection declaration is moved out of the try block to make it visible in the finally block.

```java
public static void main(String[] args) {
    Connection connection = new Connection();
    connection.open();
    try {
        // LOGIC
        String str = null;
        str.toString();

    } catch (Exception e) {
        // NOT PRINTING EXCEPTION TRACE - BAD PRACTICE
        System.out.println("Exception Handled - Main");
    } finally {
        connection.close();
    }
}
```

Output

```
Connection Opened
Exception Handled - Main
Connection Closed
```

Connection is closed even when exception is thrown. This is because connection.close() is called in the finally block.

Finally block is always executed (even when an exception is thrown). So, if we want some code to be always executed we can move it to finally block.

In what kind of scenarios, a finally block is not executed?

Code in finally is NOT executed only in two situations.

- If exception is thrown in finally.
- If JVM Crashes in between (for example, System.exit()).

Is a finally block executed even when there is a return statement in the try block?

```
private static void method2() {

        Connection connection = new Connection();
        connection.open();
        try {
            // LOGIC
            String str = null;
            str.toString();
            return;
        } catch (Exception e) {
            // NOT PRINTING EXCEPTION TRACE - BAD PRACTICE
            System.out.println("Exception Handled - Method 2");
            return;
        } finally {
            connection.close();
        }
    }
```

Is a try block without corresponding catch block allowed?

Yes. try without a catch is allowed. Example below.

```
        private static void method2() {
        Connection connection = new Connection();
        connection.open();
        try {
            // LOGIC
            String str = null;
            str.toString();
        } finally {
            connection.close();
        }
    }
```

However a try block without both catch and finally is NOT allowed.

Below method would give a Compilation Error!! (End of try block)

```
    private static void method2() {
        Connection connection = new Connection();
        connection.open();
        try {
            // LOGIC
            String str = null;
            str.toString();
        }//COMPILER ERROR!!
    }
```

Explain the hierarchy of Exception related classes in Java?

Throwable is the highest level of Error Handling classes.

Below class definitions show the pre-defined exception hierarchy in Java.

```java
//Pre-defined Java Classes
class Error extends Throwable{}
class Exception extends Throwable{}
class RuntimeException extends Exception{}
```

Below class definitions show creation of a programmer defined exception in Java.

```java
//Programmer defined classes
class CheckedException1 extends Exception{}
class CheckedException2 extends CheckedException1{}

class UnCheckedException extends RuntimeException{}
class UnCheckedException2 extends UnCheckedException{}
```

What is difference between an Error and an Exception?

Error is used in situations when there is nothing a programmer can do about an error. Ex: StackOverflowError, OutOfMemoryError. Exception is used when a programmer can handle the exception.

What is the difference between a Checked Exception and an Un-Checked Exception?

RuntimeException and classes that extend RuntimeException are called unchecked exceptions. For Example: RuntimeException,UnCheckedException,UnCheckedException2 are unchecked or RunTime Exceptions. There are subclasses of RuntimeException (which means they are subclasses of Exception also.)

Other Exception Classes (which don't fit the earlier definition). These are also called Checked Exceptions. Exception, CheckedException1,CheckedException2 are checked exceptions. They are subclasses of Exception which are not subclasses of RuntimeException.

How do you throw a Checked Exception from a Method?

Consider the example below. The method addAmounts throws a new Exception. However, it gives us a compilation error because Exception is a Checked Exception.

All classes that are not RuntimeException or subclasses of RuntimeException but extend Exception are called CheckedExceptions. The rule for CheckedExceptions is that they should be handled or thrown. Handled means it should be completed handled - i.e. not throw out of the method. Thrown means the method should declare that it throws the exception

Example without throws: Does NOT compile

```
class AmountAdder {
    static Amount addAmounts(Amount amount1, Amount amount2) {
        if (!amount1.currency.equals(amount2.currency)) {
            throw new Exception("Currencies don't match");// COMPILER ERROR!                // Unhandled exception type
Exception
        }
        return new Amount(amount1.currency, amount1.amount + amount2.amount);
    }
}
```

Example with throws definition

Let's look at how to declare throwing an exception from a method.

Look at the line "static Amount addAmounts(Amount amount1, Amount amount2) throws Exception". This is how we declare that a method throws Exception.

```
class AmountAdder {
    static Amount addAmounts(Amount amount1, Amount amount2) throws Exception {
        if (!amount1.currency.equals(amount2.currency)) {
            throw new Exception("Currencies don't match");
        }
        return new Amount(amount1.currency, amount1.amount + amount2.amount);
    }
}
```

How do you create a Custom Exception Classes?

We can create a custom exception by extending Exception class or RuntimeException class. If we extend Exception class, it will be a checked exception class. If we extend RuntimeException class, then we create an unchecked exception class.

Example

```
class CurrenciesDoNotMatchException extends Exception{
}
```

Let's now create some sample code to use CurrenciesDoNotMatchException. Since it is a checked exception we need do two things a. **throw new** CurrenciesDoNotMatchException(); b. **throws** CurrenciesDoNotMatchException (in method declaration).

```
class AmountAdder {
    static Amount addAmounts(Amount amount1, Amount amount2)
            throws CurrenciesDoNotMatchException {
        if (!amount1.currency.equals(amount2.currency)) {
            throw new CurrenciesDoNotMatchException();
        }
        return new Amount(amount1.currency, amount1.amount + amount2.amount);
    }
}
```

How should the Exception catch blocks be ordered ?

Specific Exception catch blocks should be before the catch block for a Generic Exception. For example, CurrenciesDoNotMatchException should be before Exception. Below code gives a compilation error.

```
    public static void main(String[] args) {
        try {
            AmountAdder.addAmounts(new Amount("RUPEE", 5), new Amount("DOLLAR",
                    5));
        } catch (Exception e) { // COMPILER ERROR!!
            System.out.println("Handled Exception");
        } catch (CurrenciesDoNotMatchException e) {
            System.out.println("Handled CurrenciesDoNotMatchException");
        }
    }
```

Can you explain some Exception Handling Best Practices?

Never Completely Hide Exceptions. At the least log them. printStactTrace method prints the entire stack trace when an exception occurs. If you handle an exception, it is always a good practice to log the trace.

```
    public static void main(String[] args) {
        try {
            AmountAdder.addAmounts(new Amount("RUPEE", 5), new Amount("RUPEE",
                    5));
            String string = null;
            string.toString();
        } catch (CurrenciesDoNotMatchException e) {
            System.out.println("Handled CurrenciesDoNotMatchException");
            e.printStackTrace();
        }
    }
```

Files

What are the basic methods in File class?

Create a File Object

```
File file = new File("FileName.txt");
```

Check if the file exists.

```
System.out.println(file.exists());
```

If file does not exist creates it and returns true. If file exists, returns false.

```
System.out.println(file.createNewFile());
```

Getting full path of file.

```
System.out.println(file.getAbsolutePath());
System.out.println(file.isFile());//true
System.out.println(file.isDirectory());//false
```

Renaming a file

```
File fileWithNewName = new File("NewFileName.txt");
file.renameTo(fileWithNewName);
//There is no method file.renameTo("NewFileName.txt");
```

How do you handle directories in Java?

A File class in Java represents a file and directory.

```
File directory = new File("src/com/rithus");
```

Print full directory path

```
System.out.println(directory.getAbsolutePath());
System.out.println(directory.isDirectory());//true
```

This does not create the actual file.

```
File fileInDir = new File(directory,"NewFileInDirectory.txt");
```

Actual file is created when we invoke createNewFile method.

```
System.out.println(fileInDir.createNewFile()); //true - First Time
```

Print the files and directories present in the folder.

```
System.out.println(Arrays.toString(directory.list()));
```

Creating a directory

```
File newDirectory = new File("newfolder");
System.out.println(newDirectory.mkdir());//true - First Time
```

Creating a file in a new directory

```
File notExistingDirectory = new File("notexisting");
File newFile = new File(notExistingDirectory,"newFile");

//Will throw Exception if uncommented: No such file or directory
//newFile.createNewFile();

System.out.println(newDirectory.mkdir());//true - First Time
```

How do you write to a file using FileWriter class?

We can write to a file using FileWriter class.

Write a string to a file using FileWriter

```
//FileWriter helps to write stuff into the file
FileWriter fileWriter = new FileWriter(file);
fileWriter.write("How are you doing?");
//Always flush before close. Writing to file uses Buffering.
fileWriter.flush();
fileWriter.close();
```

FileWriter Constructors

FileWriter Constructors can accept file(File) or the path to file (String) as argument. When a writer object is created, it creates the file - if it does not exist.

```
FileWriter fileWriter2 = new FileWriter("FileName.txt");
fileWriter2.write("How are you doing Buddy?");
//Always flush before close. Writing to file uses Buffering.
fileWriter2.flush();
fileWriter2.close();
```

How do you read from a file using FileReader class?

File Reader can be used to read entire content from a file at one go.

Read from file using FileReader

```
FileReader fileReader = new FileReader(file);
char[] temp = new char[25];

//fileReader reads entire file and stores it into temp
System.out.println(fileReader.read(temp));//18 - No of characters Read from file
System.out.println(Arrays.toString(temp));//output below
//[H, o, w,  , a, r, e,  , y, o, u,  , d, o, i, n, g, ?, , , , , ,]

fileReader.close();//Always close anything you opened:)
```

FileReader Constructors

FileReader constructors can accept file(File) or the path to file (String) as argument.

```
FileReader fileReader2 = new FileReader("FileName.txt");
System.out.println(fileReader2.read(temp));//24
System.out.println(Arrays.toString(temp));//output below
```

What is the use of BufferedWriter and BufferedReader classes in Java?

BufferedWriter and BufferedReader provide better buffering in addition to basic file writing and reading operations. For example, instead of reading the entire file, we can read a file line by line. Let's write an example to write and read from a file using FileReader and FileWriter.

BufferedWriter class helps writing to a class with better buffering than FileWriter. BufferedWriter Constructors only accept another Writer as argument.

```
FileWriter fileWriter3 = new FileWriter("BufferedFileName.txt");
BufferedWriter bufferedWriter = new BufferedWriter(fileWriter3);
bufferedWriter.write("How are you doing Buddy?");
bufferedWriter.newLine();
bufferedWriter.write("I'm Doing Fine");
//Always flush before close. Writing to file uses Buffering.
bufferedWriter.flush();
bufferedWriter.close();
fileWriter3.close();
```

BufferedReader helps to read the file line by line. BufferedReader Constructors only accept another Reader as argument.

```
FileReader fileReader3 = new FileReader("BufferedFileName.txt");
BufferedReader bufferedReader = new BufferedReader(fileReader3);

String line;
//readLine returns null when reading the file is completed.
while((line=bufferedReader.readLine()) != null){
    System.out.println(line);
}
```

What is the use of PrintWriter class?

PrintWriter provides advanced methods to write formatted text to the file. It supports printf function. PrintWriter constructors supports varied kinds of arguments – `File`, `String (File Path)` and `Writer`.

```
PrintWriter printWriter = new PrintWriter("PrintWriterFileName.txt");
```

Other than write function you can use format, printf, print, println functions to write to PrintWriter file.

```
//writes "        My Name" to the file
printWriter.format("%15s", "My Name");
```

```java
printWriter.println(); //New Line
printWriter.println("Some Text");

//writes "Formatted Number: 4.50000" to the file
printWriter.printf("Formatted Number: %5.5f", 4.5);
printWriter.flush();//Always flush a writer
printWriter.close();
```

Serialization

What is Serialization?

Serialization helps us to save and retrieve the state of an object.

- Serialization => Convert object state to some internal object representation.
- De-Serialization => The reverse. Convert internal representation to object.

Two important methods

- ObjectOutputStream.writeObject() // serialize and write to file
- ObjectInputStream.readObject() // read from file and deserialize

How do you serialize an object using Serializable interface?

To serialize an object it should implement Serializable interface. In the example below, Rectangle class implements Serializable interface. Note that Serializable interface does not declare any methods to be implemented.

Below example shows how an instance of an object can be serialized. We are creating a new Rectangle object and serializing it to a file Rectangle.ser.

```
class Rectangle implements Serializable {
    public Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
        area = length * breadth;
    }

    int length;
    int breadth;
    int area;
}

FileOutputStream fileStream = new FileOutputStream("Rectangle.ser");
ObjectOutputStream objectStream = new ObjectOutputStream(fileStream);
objectStream.writeObject(new Rectangle(5, 6));
objectStream.close();
```

How do you de-serialize in Java?

Below example show how a object can be deserialized from a serialized file. A rectangle object is deserialized from the file Rectangle.ser

```
FileInputStream fileInputStream = new FileInputStream("Rectangle.ser");
ObjectInputStream objectInputStream = new ObjectInputStream(
        fileInputStream);
Rectangle rectangle = (Rectangle) objectInputStream.readObject();
objectInputStream.close();
System.out.println(rectangle.length);// 5
System.out.println(rectangle.breadth);// 6
System.out.println(rectangle.area);// 30
```

What do you do if only parts of the object have to be serialized?

We mark all the properties of the object which should not be serialized as transient. Transient attributes in an object are not serialized. Area in the previous example is a calculated value. It is unnecessary to serialize and deserialize. We can calculate it when needed. In this situation, we can make the variable transient. Transient variables are not serialized. (`transient int area;`)

//Modified Rectangle class

```
class Rectangle implements Serializable {
    public Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
        area = length * breadth;
    }

    int length;
    int breadth;
    transient int area;
}
```

If you run the program again, you would get following output

```
System.out.println(rectangle.length);// 5
System.out.println(rectangle.breadth);// 6
System.out.println(rectangle.area);// 0
```

Note that the value of rectangle.area is set to 0. Variable area is marked transient. So, it is not stored into the serialized file. And when de-serialization happens area value is set to default value i.e. 0.

How do you serialize a hierarchy of objects?

Objects of one class might contain objects of other classes. When serializing and de-serializing, we might need to serialize and de-serialize entire object chain. All classes that need to be serialized have to implement the Serializable interface. Otherwise, an exception is thrown. Look at the class below. An object of class House contains an object of class Wall.

```
class House implements Serializable {
    public House(int number) {
        super();
        this.number = number;
    }

    Wall wall;
    int number;
}

class Wall{
    int length;
    int breadth;
    int color;
}
```

House implements Serializable. However, Wall doesn't implement Serializable. When we try to serialize an instance of House class, we get the following exception.

Output:

```
Exception in thread "main" java.io.NotSerializableException: com.rithus.serialization.Wall
    at java.io.ObjectOutputStream.writeObject0(Unknown Source)
    at java.io.ObjectOutputStream.defaultWriteFields(Unknown Source)
```

This is because Wall is not serializable. Two solutions are possible.

- Make Wall transient. Wall object will not be serialized. This causes the wall object state to be lost.
- Make Wall implement Serializable. Wall object will also be serialized and the state of wall object along with the house will be stored.

```java
class House implements Serializable {
    public House(int number) {
        super();
        this.number = number;
    }

    transient Wall wall;
    int number;
}

class Wall implements Serializable {
    int length;
    int breadth;
    int color;
}
```

With both these programs, earlier main method would run without throwing an exception.

If you try de-serializing, In Example2, state of wall object is retained whereas in Example1, state of wall object is lost.

Are the constructors in an object invoked when it is de-serialized?

No. When a class is De-serialized, initialization (constructor’s, initializer’s) does not take place. The state of the object is retained as it is.

Are the values of static variables stored when an object is serialized?

Static Variables are not part of the object. They are not serialized.

Multi Threading

What is the need for Threads in Java?

Threads allow Java code to run in parallel. Let’s look at an example to understand what we can do with Threads.

Need for Threads

We are creating a Cricket Statistics Application. Let's say the steps involved in the application are

- STEP I: Download and Store Bowling Statistics => 60 Minutes
- STEP II: Download and Store Batting Statistics => 60 Minutes
- STEP III: Download and Store Fielding Statistics => 15 Minutes
- STEP IV: Merge and Analyze => 25 Minutes

Steps I, II and III are independent and can be run in parallel to each other. Run individually this program takes 160 minutes. We would want to run this program in lesser time. Threads can be a solution to this problem. Threads allow us to run STEP I, II and III in parallel and run Step IV when all Steps I, II and III are completed.

Below example shows the way we would write code usually – without using Threads.

```
ThreadExamples example = new ThreadExamples();
```

```
example.downloadAndStoreBattingStatistics();
example.downloadAndStoreBowlingStatistics();
example.downloadAndStoreFieldingStatistics();

example.mergeAndAnalyze();
```

downloadAndStoreBowlingStatistics starts only after downloadAndStoreBattingStatistics completes execution. downloadAndStoreFieldingStatistics starts only after downloadAndStoreBowlingStatistics completes execution. What if I want to run them in parallel without waiting for the others to complete?

This is where Threads come into picture. Using Multi-Threading we can run each of the above steps in parallel and synchronize when needed. We will understand more about synchronization later.

How do you create a thread?

Creating a Thread class in Java can be done in two ways. Extending Thread class and implementing Runnable interface. Let's create the BattingStatisticsThread extending Thread class and BowlingStatisticsThread implementing Runnable interface.

How do you create a thread by extending Thread class?

Thread class can be created by extending Thread class and implementing the public void run() method.

Look at the example below: A dummy implementation for BattingStatistics is provided which counts from 1 to 1000.

```java
class BattingStatisticsThread extends Thread {
    //run method without parameters
    public void run() {
        for (int i = 0; i < 1000; i++)
            System.out
            .println("Running Batting Statistics Thread "
                            + i);
    }
}
```

How do you create a thread by implementing Runnable interface?

Thread class can also be created by implementing Runnable interface and implementing the method declared in Runnable interface "public void run()". Example below shows the Batting Statistics Thread implemented by implementing Runnable interface.

```java
class BowlingStatisticsThread implements Runnable {
    //run method without parameters
    public void run() {
        for (int i = 0; i < 1000; i++)
            System.out
            .println("Running Bowling Statistics Thread "
                              + i);
    }
}
```

How do you run a Thread in Java?

Running a Thread in Java is slightly different based on the approach used to create the thread.

Thread created Extending Thread class

When using inheritance, An object of the thread needs be created and start() method on the thread needs to be called. Remember that the method that needs to be called is not run() but it is start().

```
BattingStatisticsThread battingThread1 = new BattingStatisticsThread();
battingThread1.start();
```

Thread created implementing RunnableInterface.

Three steps involved.

- Create an object of the BowlingStatisticsThread(class implementing Runnable).
- Create a Thread object with the earlier object as constructor argument.
- Call the start method on the thread.

```
BowlingStatisticsThread battingInterfaceImpl = new BowlingStatisticsThread();
Thread battingThread2 = new Thread(
        battingInterfaceImpl);
battingThread2.start();
```

What are the different states of a Thread?

Different states that a thread can be in are defined the class State.

- NEW;
- RUNNABLE;
- RUNNING;
- BLOCKED/WAITING;
- TERMINATED/DEAD;

Let's consider the example that we discussed earlier.

Example Program

```
LINE 1: BattingStatisticsThread battingThread1 = new BattingStatisticsThread();
LINE 2: battingThread1.start();

LINE 3: BowlingStatisticsThread battingInterfaceImpl = new BowlingStatisticsThread();
LINE 4: Thread battingThread2 = new Thread(battingInterfaceImpl);
LINE 5:battingThread2.start();
```

Description

A thread is in NEW state when an object of the thread is created but the start method is not yet called. At the end of line 1, `battingThread1 is in NEW state.`

A thread is in RUNNABLE state when it is eligible to run, but not running yet. (A number of Threads can be in RUNNABLE state. Scheduler selects which Thread to move to RUNNING state). In the above example, sometimes the Batting Statistics thread is running and at other time, the Bowling Statistics Thread is running. When Batting Statistics thread is Running, the Bowling Statistics thread is ready to run. It's just that the scheduler picked Batting Statistics thread to run at that instance and vice-versa. When Batting Statistics thread is Running, the Bowling Statistics Thread is in Runnable state (Note that the Bowling Statistics Thread is not waiting for anything except for the Scheduler to pick it up and run it).

A thread is RUNNING state when it's the one that is currently , what else to say, Running.

A thread is in BLOCKED/WAITING/SLEEPING state when it is not eligible to be run by the Scheduler. Thread is alive but is waiting for something. An example can be a Synchronized block. If Thread1 enters synchronized block, it blocks all the other threads from entering synchronized code on the same instance or class. All other threads are said to be in Blocked state.

A thread is in DEAD/TERMINATED state when it has completed its execution. Once a thread enters dead state, it cannot be made active again.

What is priority of a thread? How do you change the priority of a thread?

Scheduler can be requested to allot more CPU to a thread by increasing the threads priority. Each thread in Java is assigned a default Priority 5. This priority can be increased or decreased (Range 1 to 10).

If two threads are waiting, the scheduler picks the thread with highest priority to be run. If all threads have equal priority, the scheduler then picks one of them randomly. Design programs so that they don't depend on priority.

Thread Priority Example

Consider the thread example declared below:

```java
class ThreadExample extends Thread {
    public void run() {
        for (int i = 0; i < 1000; i++)
            System.out
                    .println( this.getName() + " Running "
                            + i);
    }
}
```

Priority of thread can be changed by invoking setPriority method on the thread.

```java
ThreadExample thread1 = new ThreadExample();
thread1.setPriority(8);
```

Java also provides predefined constants Thread.MAX_PRIORITY(10), Thread.MIN_PRIORITY(1), Thread.NORM_PRIORITY(5) which can be used to assign priority to a thread.

Synchronization

What is synchronization of threads?

Since Threads run in parallel, a new problem arises. What if thread1 modifies data which is being accessed by thread2? How do we ensure that different threads don't leave the system in an inconsistent state? This problem is usually called synchronization problem.

Let's first look at an example where this problem can occur. Consider the code in the setAndGetSum method.

```
int setandGetSum(int a1, int a2, int a3) {
    cell1 = a1;
    sleepForSomeTime();
    cell2 = a2;
    sleepForSomeTime();
    cell3 = a3;
    sleepForSomeTime();
    return cell1 + cell2 + cell3;
}
```

If following method is running in two different threads, funny things can happen. After setting the value to each cell, there is a call for the Thread to sleep for some time. After Thread 1 sets the value of cell1, it goes to Sleep. So, Thread2 starts executing. If Thread 2 is executing "**return** cell1 + cell2 + cell3;", it uses cell1 value set by Thread 1 and cell2 and cell3 values set by Thread 2. This results in the unexpected results that we see when the method is run in parallel. What is explained is one possible scenario. There are several such scenarios possible.

The way you can prevent multiple threads from executing the same method is by using the synchronized keyword on the method. If a method is marked synchronized, a different thread gets access to the method only when there is no other thread currently executing the method.

Let's mark the method as synchronized:

```
synchronized int setandGetSum(int a1, int a2, int a3) {
    cell1 = a1;
    sleepForSomeTime();
    cell2 = a2;
    sleepForSomeTime();
    cell3 = a3;
    sleepForSomeTime();
    return cell1 + cell2 + cell3;
}
```

Can you give an example of a synchronized block?

All code which goes into the block is synchronized on the current object.

```
    void synchronizedExample2() {
        synchronized (this){
        //All code goes here..
        }
    }
```

Can a static method be synchronized?

Yes. Consider the example below.

```
    synchronized static int getCount(){
        return count;
    }
```

Static methods and block are synchronized on the class. Instance methods and blocks are synchronized on the instance of the class i.e. an object of the class. Static synchronized methods and instance synchronized methods don't affect each other. This is because they are synchronized on two different things.

```
    static int getCount2(){
        synchronized (SynchronizedSyntaxExample.class) {
            return count;
        }
    }
```

What is the use of join method in threads?

Join method is an instance method on the Thread class. Let's see a small example to understand what join method does.

Let's consider the thread's declared below: thread2, thread3, thread4

```
ThreadExample thread2 = new ThreadExample();
ThreadExample thread3 = new ThreadExample();
ThreadExample thread4 = new ThreadExample();
```

Let’s say we would want to run thread2 and thread3 in parallel but thread4 can only run when thread3 is finished. This can be achieved using join method.

Join method example

Look at the example code below:

```
thread3.start();
thread2.start();
thread3.join();//wait for thread 3 to complete
System.out.println("Thread3 is completed.");
thread4.start();
```

thread3.join() method call force the execution of main method to stop until thread3 completes execution. After that, thread4.start() method is invoked, putting thread4 into a Runnable State.

Overloaded Join method

Join method also has an overloaded method accepting time in milliseconds as a parameter.

```
thread4.join(2000);
```

In above example, main method thread would wait for 2000 ms or the end of execution of thread4, whichever is minimum.

Describe a few other important methods in Threads?

Thread yield method

Yield is a static method in the Thread class. It is like a thread saying " I have enough time in the limelight. Can some other thread run next?".

A call to yield method changes the state of thread from RUNNING to RUNNABLE. However, the scheduler might pick up the same thread to run again, especially if it is the thread with highest priority.

Summary is yield method is a request from a thread to go to Runnable state. However, the scheduler can immediately put the thread back to RUNNING state.

Thread sleep method

sleep is a static method in Thread class. sleep method can throw a InterruptedException. sleep method causes the thread in execution to go to sleep for specified number of milliseconds.

What is a deadlock?

Let's consider a situation where thread1 is waiting for thread2 (thread1 needs an object whose synchronized code is being executed by thread1) and thread2 is waiting for thread1. This situation is called a Deadlock. In a Deadlock situation, both these threads would wait for one another for ever.

What are the important methods in java for inter-thread communication?

Important methods are wait, notify and notifyAll.

What is the use of wait method?

Below snippet shows how wait is used. wait method is defined in the Object class. This causes the thread to wait until it is notified.

```
synchronized(thread){
    thread.start();
    thread.wait();
}
```

What is the use of notify method?

Below snippet shows how notify is used. notify method is defined in the Object class. This causes the object to notify other waiting threads.

```
        synchronized (this) {
         calculateSumUptoMillion();
         notify();
    }
```

What is the use of notifyAll method?

If more than one thread is waiting for an object, we can notify all the threads by using notifyAll method.

thread.notifyAll();

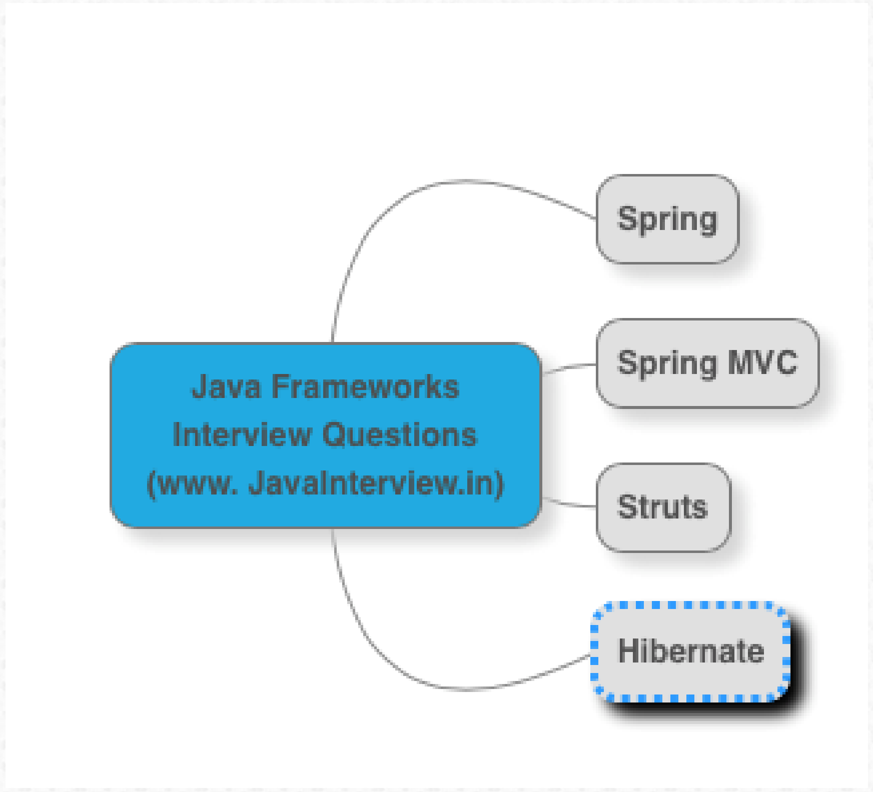
Can you write a synchronized program with wait and notify methods?

```
package com.rithus.threads;

class Calculator extends Thread {
    long sumUptoMillion;
    long sumUptoTenMillion;

    public void run() {
        synchronized (this) {
            calculateSumUptoMillion();
            notify();
        }
        calculateSumUptoTenMillion();
    }

    private void calculateSumUptoMillion() {
        for (int i = 0; i < 1000000; i++) {
            sumUptoMillion += i;
        }
        System.out.println("Million done");
    }

    private void calculateSumUptoTenMillion() {
        for (int i = 0; i < 10000000; i++) {
            sumUptoTenMillion += i;
        }
        System.out.println("Ten Million done");
    }
}

public class ThreadWaitAndNotify {
    public static void main(String[] args) throws InterruptedException {
        Calculator thread = new Calculator();
        synchronized(thread){
            thread.start();
            thread.wait();
        }
        System.out.println(thread.sumUptoMillion);
    }
}
```

Output

```
Million done
499999500000
Ten Million done
```

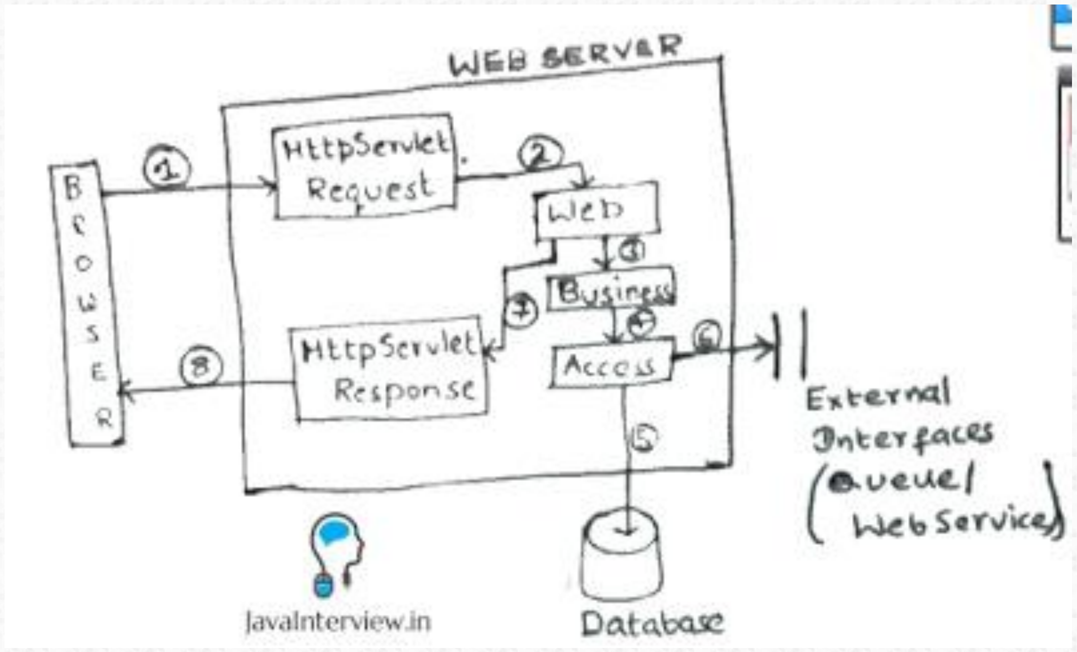
Web Applications and MVC



How do traditional web applications work?

Traditional web applications are based on HTTP Request and HTTP Response cycle. Following are the steps:

- When user initiates an action in the browser, A HTTP Request is created by the browser.
- Web Server creates a HTTPServletRequest based on the content of HTTP Request.
- The web application (based on the framework used) handles the HTTPServletRequest. (Controllers, Business Layer, database calls and external interfaces)
- A HTTPServletResponse is returned. This is converted to HTTP Response.
- The HTTP Response is rendered by the browser

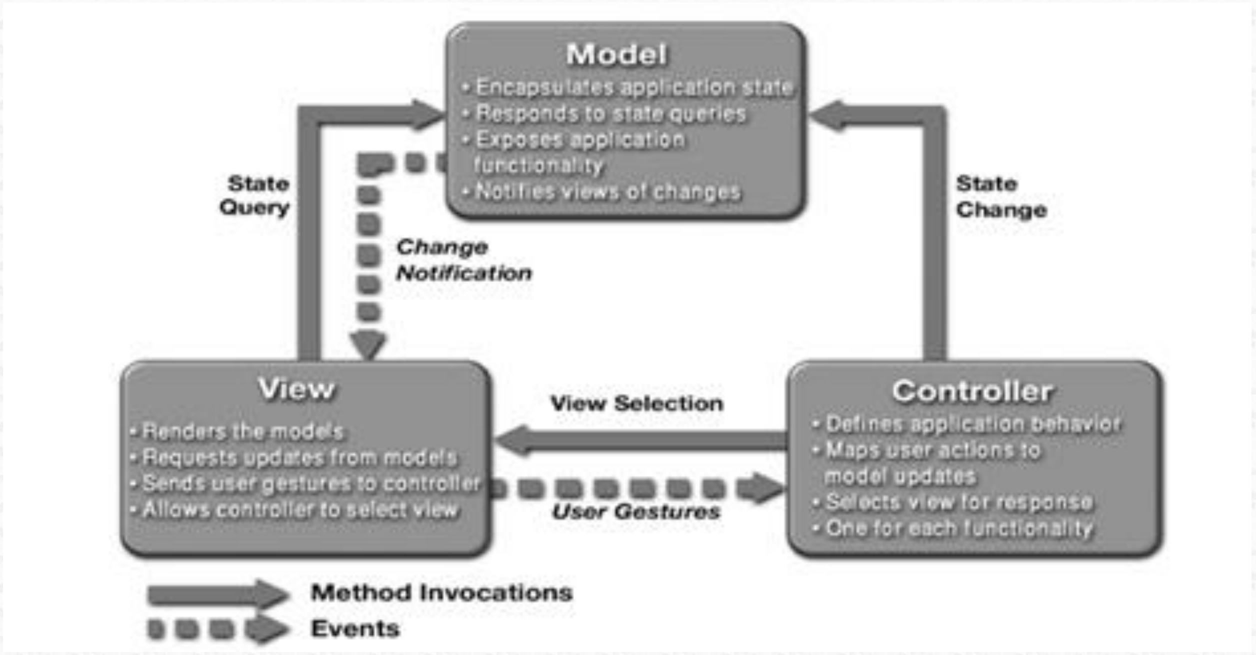


What is MVC Pattern?

MVC stands for Model, View and Controller. It is a software architectural pattern for implementing user interfaces.

- Controller : Controls the flow. Sends commands to the model to update the model's state. Sends commands to view to change the view's presentation of the model.
- Model : Represents the state of the application. Sometimes - notifies associated views and controllers when there is a change in its state.

- View : Visual representation of the model for the user.

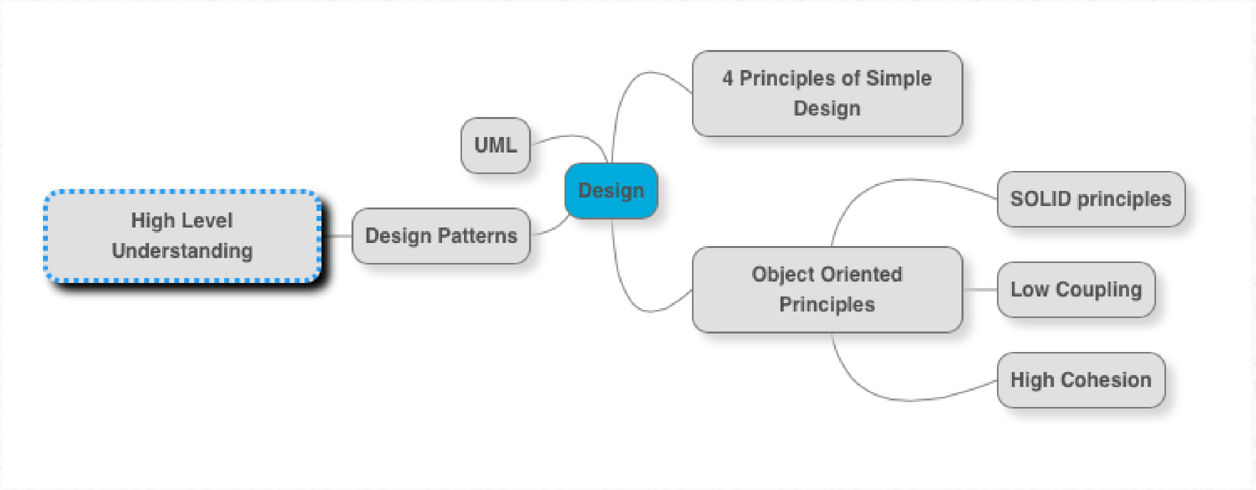

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

For answers to all questions in this Section in a Video Format - buy our Complete Interview Guide – 24 Videos & 400+ Questions – 50% Discount

Now available at 50% Discount - Rs 799 / $ 12.99 only. Visit ***http://www.in28minutes.com/p/buy-our-java-interview-guide.html***

Design

As an programmer, what are design principles you focus on?



I start off with the 4 Principles of Simple Design. Following playlist explains the four principles of simple design in detail : https://www.youtube.com/watch?v=OwS8ydVTx1c&list=PL066F8F24976D837C

- Runs all tests
- Minimize Duplication
- Maximize Clarity

- Keep it Small

Next important design principles would be those related to Object Oriented Programming. Have good object, which have well-defined responsibilities. Following are the important concepts you need to have a good overview of. These are covered in various parts in the video https://www.youtube.com/watch?v=0xcgzUdTO5M. Also, look up the specific videos for each topic.

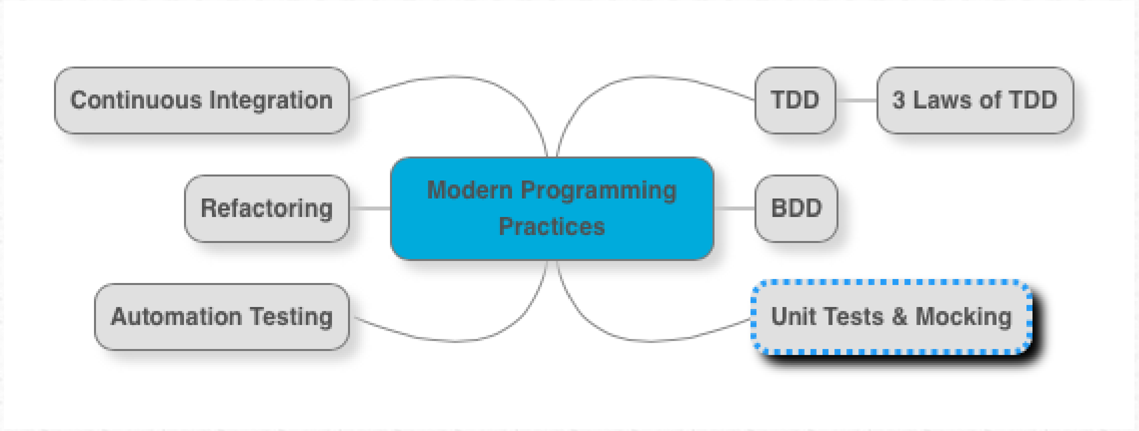
- Coupling
- Cohesion : https://www.youtube.com/watch?v=BkcQWoF5124&list=PLBBog2r6uMCTJ5JLyOySaOTrYdpWq48vK&index=9
- Encapsulation
- Polymorphism : https://www.youtube.com/watch?v=t8PTatUXtpI&list=PL91AF2D4024AA59AF&index=5
- SOLID Principles

UML is next even though, formal use of UML is on the way down with Agile. However, I think UML is a great tool in the arsenal for a white board discussion on design. A picture is worth thousand words. I recommend having a good overview of the UML basics. Focus on these four before you move on to others.

- Class diagrams
- Sequence diagrams
- Component diagrams
- Deployment diagrams

Last and also the least important is Design Patterns. Following video covers all the major design patterns. https://www.youtube.com/watch?v=0jjNjXcYmAU. My personal view : Design Patterns are good to know. Have a good idea on what each one of them does. But, that where it ends. I'm not a big fan of understanding the intricate details of each Design Pattern. You can look it up if you have a good overall idea about Design Patterns.

What are the modern programming practices which lead to very good applications?



First of all : Unit Testing and Mocking. We are in the age of continuous integration and delivery, and the basic thing that enables those is having a good set of unit test in place. (Don't confuse unit testing with screen testing done manually to check if the screen flow is right. What I mean by unit testing is JUnit test's checking the business logic/screen flow in a java method (or) set of methods). Understand JUnit. Here is a good start : https://www.youtube.com/watch?v=AN4NCnc4eZg&list=PL83C941BB0D27A6AF. Also understand the concept of Mocking. When should we mock? And when we should not? Complicated question indeed. Understand atleast one mocking framework : Mockito is the most popular one. Easymock is a good mocking framework as well.

Second in line is Automated Integration Tests. Automated Integration Tests is the second important bullet in enabling continuous delivery. Understand Fitnesse, Cucumber and Protractor.

Third is TDD (actually I wanted to put it first). Understand what TDD is. If you have never used TDD, then be ready for a rude shock. Its not easy to change a routine you developed during decades (or years) of programming. Once you are used to TDD you never go back. I promise. This list of videos is a good start to understanding TDD. https://www.youtube.com/watch?v=xubiP8WoT4E&list=PLBD6D61C0A9F671F6. Have fun.

Next comes BDD. In my experience, I found BDD a great tool to enable communication between the ready team (Business Analysts, Product Owner) and the done team (Developers, Testers, Operations). When User Stories are nothing but a set of scenarios specified is GWT (Given When Then) format, it is easy for the done team to chew at the user story scenario by scenario. With tools like Cucumber & Fitnesse, tooling is not far behind too. Do check BDD out.

Next in line is Refactoring. IUnderstand refactoring. Understand the role of automation tests in refactoring.

Last (but not the least) in the line is Continuous Integration. Every project today has continuous integration. But, the real question is "What is under Continuous Integration?". Compilation, unit tests and code quality gate(s) is the bare minimum. If you have integration and chain tests, wonderful. But make sure the build does not take long. Immediate feedback is important. If needed, create a separate build scheduled less frequently for slower tests (integration and chain tests). Jenkins is the most popular Continuous Integration tool today.

What are the typical things you would need to consider while designing the Business Layer of a Java EE Web Application?

Listed below are some of the important considerations

- Should I have a Service layer acting as a facade to the Business Layer?
- How do I implement Transaction Management? JTA or Spring based Transactions or Container Managed Transactions? What would mark the boundary of transactions. Would it be service facade method call?
- Can (Should) I separate any of the Business Logic into seperate component or service?
- Do I use a Domain Object Model?
- Do I need caching? If so, at what level?
- Does service layer need to handle all exceptions? Or shall we leave it to the web layer?
- Are there any Operations specific logging or auditing that is needed?Can we implement it as a cross cutting concern using AOP?
- Do we need to validate the data that is coming into the Business Layer? Or is the validation done by the web layer sufficient?

What are the things that you would need to consider when designing the Access Layer (Data Layer) of the web application?

- Do we want to use a JPA based object mapping framework (Hibernate) or query based mapping framework (iBatis) or simple Spring DO?

- How do we communicate with external systems? Web services or JMS? If web services, then how do we handle object xml mapping? JAXB or XMLBeans?
- How do you handle connections to Database? These days, its an easy answer : leave it to the application server configuration of Data Source.
- What are the kinds of exceptions that you want to throw to Business Layer? Should they be checked exceptions or unchecked exceptions?
- Ensure that Performance and Scalability is taken care of in all the decisions you make.

What are the things that you would need to consider when designing the Web Layer?

- First question is do we want to use a modern front end javascript framework like AngularJS? If the answer is yes, most of this discussion does not apply. If the answer is no, then proceed?
- Should we use a MVC framework like Spring MVC,Struts or should we go for a Java based framework like Wicket or Vaadin?
- What should be the view technology? JSP, JSF or Template Based (Velocity, Freemarker)?
- Do you want AJAX functionality?
- How do you map view objects to business objects and vice-versa? Do you want to have View Assemblers and Business Assemblers?

- What kind of data is allowed to be put in user session? Do we need additional control mechanisms to ensure session size is small as possible?
- How do we Authenticate and Authorize users? Do we need to integrated external frameworks like Spring Security?
- Do we need to expose external web services?

What are the important features of IDE Eclipse?

Go through this youtube playlist. It takes you through all the important features of Eclipse:

https://www.youtube.com/watch?v=nbyR_M0L-vg&list=PLBBog2r6uMCSmPLJMkXXa0lgMFwGScAP8&index=1

What are the best practices for build tool Maven?

Use archetypes as much as possible. Archetypes are good start for generating projects (lookup : mvn archetype:generate) based on Spring, Spring MVC, Struts, Hibernate and a wide variety of other projects. Also, it is a good practice to create maven archetype for the components we create repeatedly (access components, consuming/exposing web services).

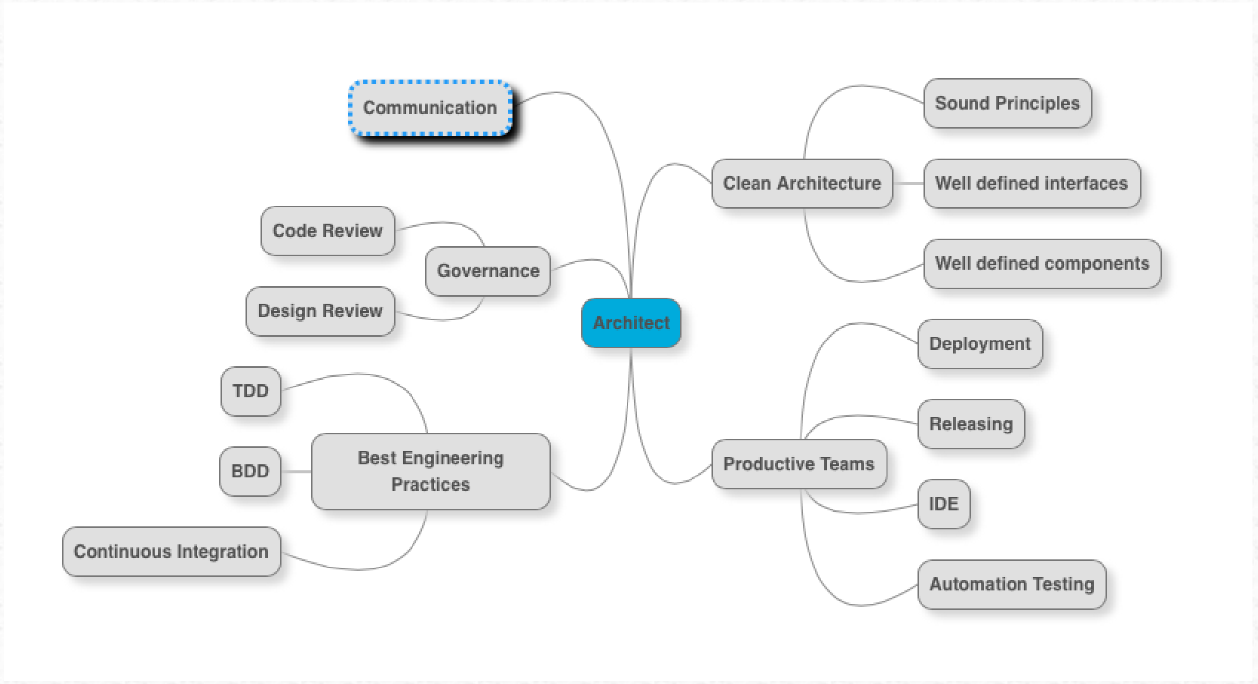
Some of the Maven Best Practices are

- Proper Dependency Mgmt : Version for one dependency at one place - preferably in the parent pom.

- Group related dependencies.
- Exclude test dependencies from final ear.
- Have a parent pom.
- Use Profiles as needed.

Architect

What are the important responsibilities of Architect?

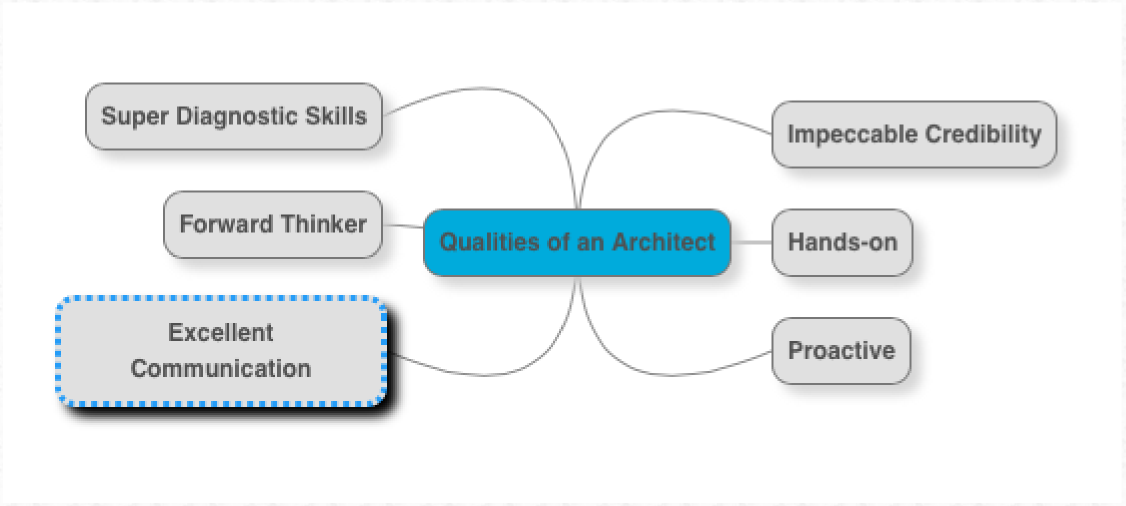


Below is a high level summary

- Creating a clean architecture based on sound principles. Architecture covering all Non Functional Requirements.
- Having good governance in place. Good review processes in place for Architecture, Design and Code.

- Ensuring teams are as productive as they can be. Right tools.
- Ensuring teams are following the best engineering practices.
- Ensuring clear communication about architecture with business and technical teams.

How should an ideal architect be like?

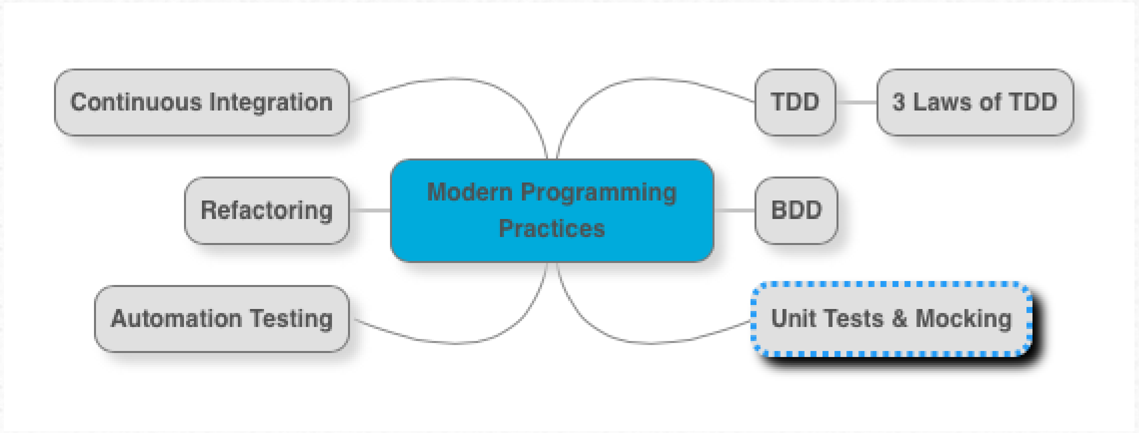


Most important qualities I look for in an Architect are

- Impeccable Credibility : Somebody the team looks up to and aspires to be.
- Super diagnostic skills : The ability to do a deep dive on a technology issue. When developers are struggling with a problem (having tried different things), Can he/she provide a fresh pair of eyes to look at the same problem?

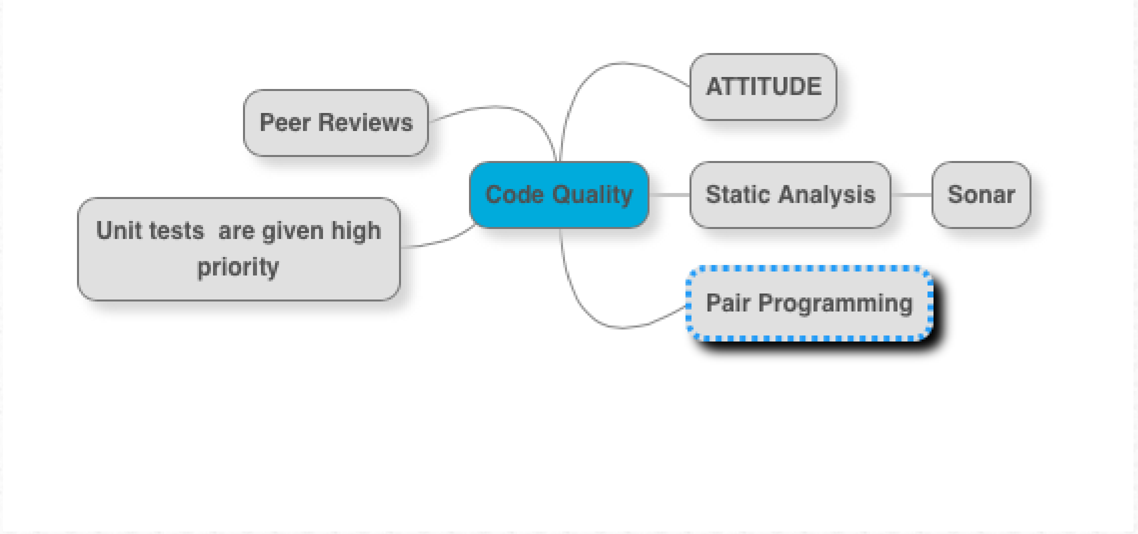
- Forward Thinker and Proactive : Never satisfied with where we are. Identifies opportunities to add value fast.
- *Great Communication* : Communication in the widest sense. Communicating the technical aspects to the stakeholders, project management, software developers, testers, etc.

What are the modern programming practices an architect should be aware of?



For more details refer to design interview questions.

How do you ensure that the Code Quality is maintained?



More than everything else, code quality is an attitude. Either, the team has it or not. The attitude to refactor when something is wrong. The attitude to be a boy scout. As an architect, it is important to create an environment where such an attitude is appreciated. (There are always bad sheep, who take the code quality to such depth that it is not fun anymore, but I like them more than developers who keep churning out bad code).

Have a good static analysis tool(and is part of Continuous Integration). Sonar is the best bet today. Understand the limits of Static Analysis. Static Analysis is not a magic wand. For me, results from Static Analysis are a signal: It helps me decide where I should look during peer or architect reviews?

Have good peer review practices in place. Every user story has to be peer reviewed. Put more focus on peer reviews when there is a new developer or there is a new technical change being done. Make best use of Pair Programming. The debate is ongoing : Is pair programming more productive or not? I would rather stay out of it. You make your choice. However, these two scenarios are bare minimum:

- Onboarding a new programmer. Makes him comfortable with all the new things he has to encounter.
- Implementing a complex functionality.

Next question is how to approach a Code Review. Difficult to cover everything. I would make a start. When doing a code review, I start with static analysis results (for example, sonar). I spend 10 minutes getting an overview of components and/or layers (focusing on size and dependencies). Next I would pick up a unit test for a complex functionality. I feel unit tests are the best place to discover the dependencies and naming practices (I believe good names = 50% of maintainable code). If a programmer can write a simple and understandable unit test, he can definitely write good code. Next, I look for 4 principles of Simple Design. After this, there are 100 other things we can look for - You decide.

How do Agile and Architecture go hand in hand?

First of all I'm a great believer that agile and architecture go hand in hand. I do not believe agile means no architecture. I think agile brings in the need to separate architecture and design. For me architecture is about things which are difficult to change : technology choices, framework choices, communication between systems etc. It would be great if a big chunk of architectural decisions are made before the done team starts. There would always be things which are uncertain. Inputs to these can come from spikes that are done as part of the Done Scrum Team.But these should be planned ahead.

Architecture choices should be well thought out. Its good to spend some time to think (Sprint Zero) before you make a architectural choice.

I think most important part of Agile Architecture is Automation Testing. Change is continuous only when the team is sure nothing is broken. And automation test suites play a great role in providing immediate feedback.

Important principles for me are test early, fail fast and automate.

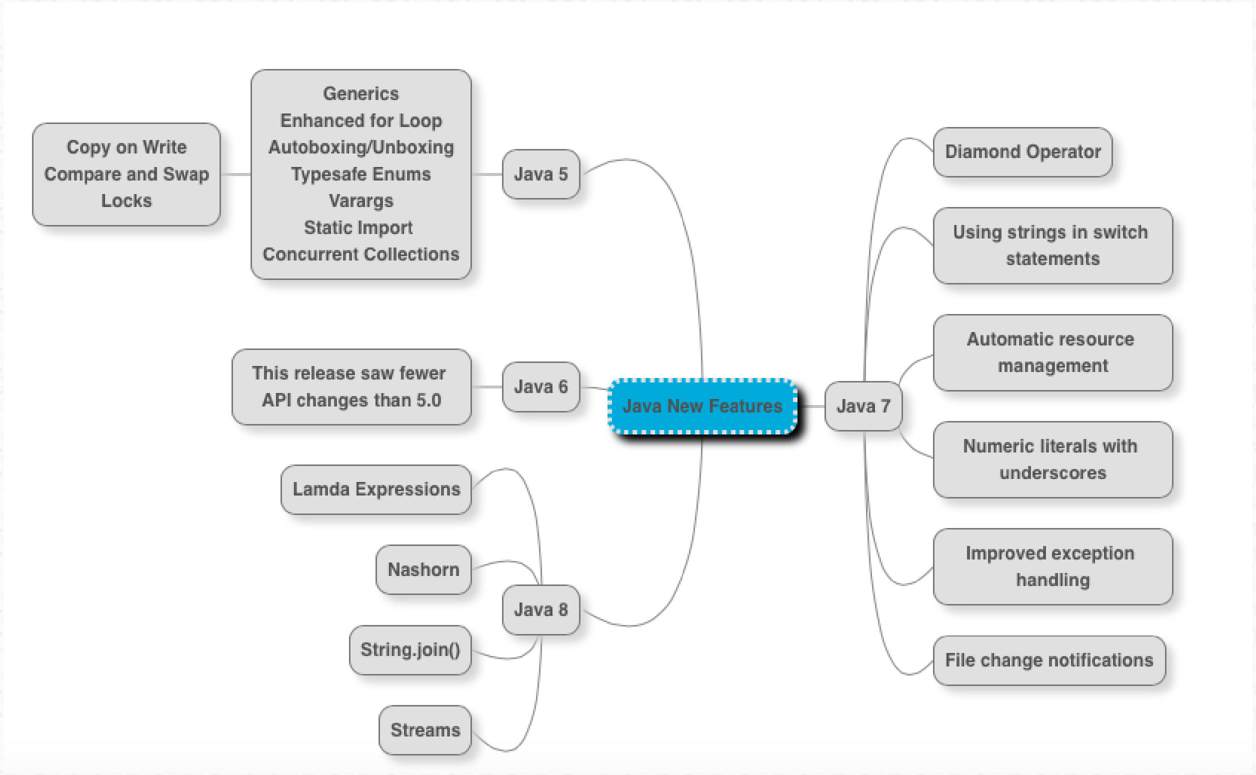
How do you ensure the team is following sound engineering practices?

I ask the following questions:

- How often is code committed?
- How often is code released?
- How often do builds break? Are they immediately fixed?
- How often is code deployed?
- What steps are part of continuous integration build? Is deployment and automation suite part of it?
- Does the team develop vertical slices when implementing a new functionality?

More questions are covered in the section on Design Interview Questions.

Java New Features



What are new features in Java 5?

New features in Java 5 are :

- Generics
- Enhanced for Loop

- Autoboxing/Unboxing
- Typesafe Enums
- Varargs
- Static Import
- Concurrent Collections
- Copy on Write
- Compare and Swap
- Locks

For more details about these new features, refer Core and Advanced Java Interview Questions.

What are the new features in Java 6?

Java 6 has very few important changes in terms of api's. There are a few performance improvements but none significant enough to deserve a mention.

What are the new features in Java 7?

New features in Java 7 are :

- Diamond Operator. Example : Map<String, List<Trade>> trades = new TreeMap <> ();

- Using String in switch statements
- Automatic resource management : try(resources_to_be_cleant){ // your code }
- Numeric literals with underscores
- Improved exception handling : Multiple catches in same block-catch(ExceptionOne | ExceptionTwo | ExceptionThree e)
- File change notifications

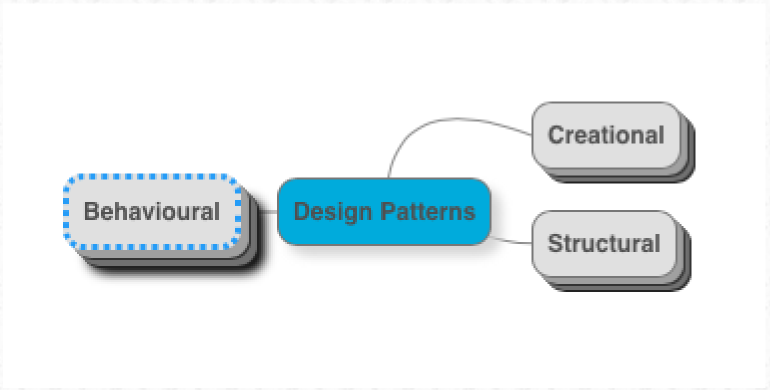
What are the new features in Java 8?

New features in Java 8 are :

- Lamda Expressions. Example : Runnable java8Runner = () -> { sop("I am running"); };
- Nashorn : javascript engine that enables us to run javascript to run on a jvm
- String.join() function
- Streams

Design Patterns

Idea behind this article is to give an overview of Design Patterns and not really explain all the implementation details related to them. For me, understanding the basics of a design pattern is important. The implementation details are secondary. Implementation details can easily be looked up when needed if I understand the context in which a design pattern applies.



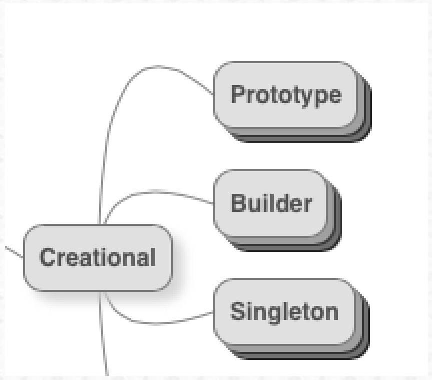
There are three different types of Design Patterns

- Creational Patterns : Concerned with creation of Objects. Prototype, Singleton, Builder etc.
- Structural Patterns : Concerned with structure of Objects and the relationships between them. Decorator, Facade, Adapter etc.

- Behavioural Patterns : Concerned with interaction between objects. Strategy, Template Method etc.

Creational Patterns

Creational Patterns deal with creation of objects.



Prototype

Prototype is defined as "A fully initialized instance to be copied or cloned". Let's consider a Chess Game. At the start of the game, the way the pieces are arranged on a board is the same. Following strategy can be used in a chess program to setup a new game:

Create a fully initialized instance of a chess game with the correct positions for all pieces.This is the prototype.

Whenever there is a need to create a new chess game, clone or copy the initial chess game prototype.

Builder

Builder pattern is usually used to hide the complexity of an object construction. Certain objects might have complex internal structure. Every time an instance is created, the entire structure needs to be created. It is a good practice to hide this complexity from the dependant objects. And that's where the Builder pattern comes in.

Builder pattern is defined as "Separates object construction from its representation".

Example : Consider a fast-food restaurant offering a Vegetarian Meal and a Non Vegetarian Meal. A typical meal is a burger and a cold drink. Depending on the type of the meal, the burger is a vegetarian burger or a chicken burger. The drink is either an orange juice or a pineapple juice. The below Builder class can be used to create meal objects.

```
public class MealBuilder {
  public Meal buildVegMeal (){
    Meal meal = new Meal();
    meal.addItem(new VegetarianBurger());
```

```
      meal.addItem(new OrangeJuice());
      return meal;
   }

   public Meal buildNonVegMeal (){
      Meal meal = new Meal();
      meal.addItem(new ChickenBurger());
      meal.addItem(new PineappleJuice());
      return meal;
   }
}
```

Singleton

Singleton is defined as "A class of which only a single instance can exist ***(in a jvm)***. A good example of Singleton class in Java is java.lang.System.

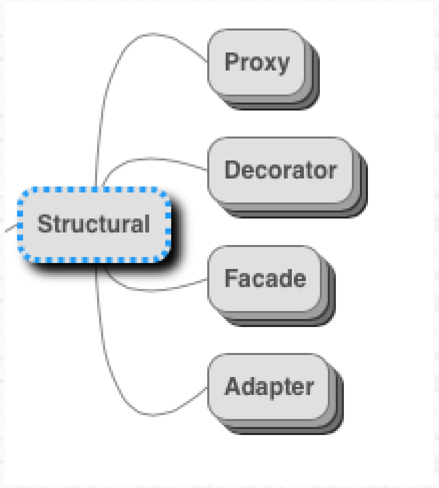
If you are a Java guys, then these things might be useful:

- Best way to implement Singleton is using a Enum. Refer "Effective Java" by Joshua Bloch.

- JEE7 has inbuilt @Singleton annotation with @Startup, @PostConstruct and @DependsOn("other beans") options
- Singletons make code difficult to unit test.
- In Spring, all beans are singletons by default (in the scope of application context).

Structural

Structural patterns deal with the structure of objects and their relationships.



Proxy

Proxy is defined as "An object representing another object".

A good example of a proxy is a Debit Card. It represents the bank account but is really not the bank account itself. It acts as a proxy to our Bank Account.

EJB's typically have two interfaces - Remote and Home. Home interface is a good example of a proxy.

Decorator

Decorator is defined as "Add responsibilities to objects dynamically".

Let's take an example of a Pizza shop offering 10 types of Pizzas. Currently these are defined using an inheritance chain. All these 10 pizza types extend AbstractPizza class. Now, there is a new functionality requested - we would want to allow 3 different types of toppings for each of the pizza type. We have two options

- Create 3 classes for each of the 10 pizza types resulting in a total of 30 classes.
- Use a topping decorator on top of AbstractPizza class.

Usually the preference would be to use a decorator.

Good example for decorator is the Java IO Class Structure. To create a LineNumberInputStream we do something like LineNumberInputStream(BufferedInputStream(FileInputStream))).

BufferedInputStream is a decorator for FileInputStream. LineNumberInputStream is a decorator for BufferedInputStream.

Disadvantage of Decorator is the resulting complexity in creating objects.

Facade

Facade is defined as “A single class that represents an entire subsystem”.

A good example of facade in real life is an event manager. We approach an event manager to organize an event and they would take care of arranging everything related to the event - Decoration, Food, Invitations, Music Band etc.

Let’s consider an application taking online orders for books. Following steps are involved.

- Check if there is stock and reserve the book.
- Make payment.
- Update stock
- Generate invoice

Its preferred to create a facade called OrderService which takes care of all these steps. Facade also become a good place to implement transaction management.

Advantages of a Facade:

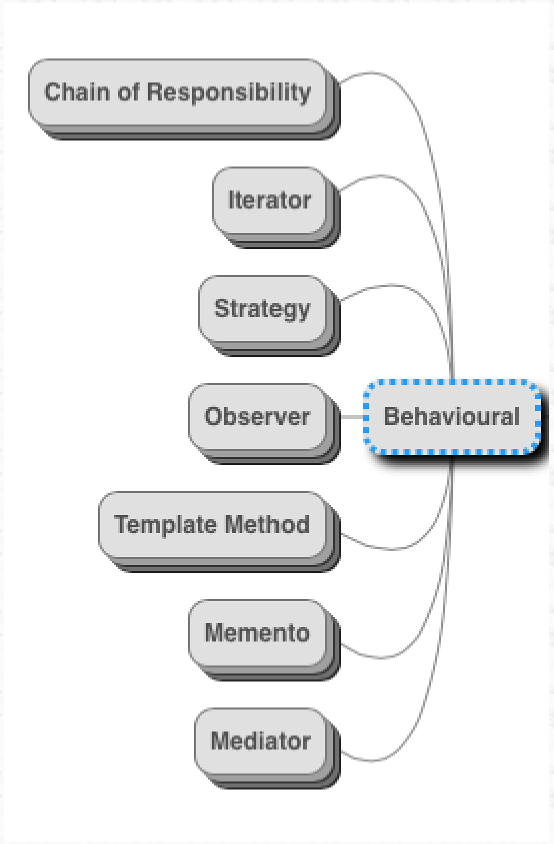
- Reduced network calls
- Reduced coupling
- Helps in establishing transaction boundary

Adapter

Defined as "Match interfaces of different classes". All the translators that we create to map one type of object to another type are good examples.

A good real life example would be Power Adapters.

Behavioural



Chain of Responsibility

Defined as “A way of passing a request between a chain of objects”.

A good real time example is the Loan or Leave Approval Process. When a loan approval is needed, it first goes to the clerk. If he cannot handle it (large amount), it goes to his manager and so on until it is approved or rejected.

Another good example is Exception Handling in most programming languages. When an exception is thrown from a method with no exception handling, it is thrown to the calling method. If there is no exception handling in that method too, it is further thrown up to its calling method and so on. This happens until an appropriate exception handler is found.

Iterator

Defined as “Sequentially access the elements of a collection”.

Different objects might have different internal representations. Iterator defines one way of looping through the objects in a list or a collection or a map, so that the internal complexities are hidden.

Strategy

Defined as “Encapsulates an algorithm inside a class”.

Typically used to decouple the algorithm or strategy used from the implementation of the class so that the algorithm can be changed independent of the class.

A good example in Java is the Comparator interface. java.util.Comparator#compare()

Observer

Defined as “A way of notifying change to a number of classes”.

A good example is Online Bidding. Different people can register as observers. They all are notified when there is a new bid.

If you are a Java programmer, Observer design pattern is already built for you. Look up the Observer interface and Observable class.

Template Method

Defined as “Defer the exact steps of an algorithm to a subclass”.

Good example is a House Plan. A high level floor plan is provided first. Within a floor plan, the foundation, framing, plumbing, and wiring will be identical for each house. Variations for each house can be added in later - additional wing, wooden flooring/carpet, which color to paint.

Another example for Template Method is implementation of Spring AbstractController. The total flow is implemented by handleRequest method.

However, subclasses can control the details by implementing the abstract method handleRequestInternal. *(example simplified to focus only on necessary details)*

```
@Override
public ModelAndView handleRequest(***) throws Exception {
        // Delegate to WebContentGenerator for checking and preparing.
        checkAndPrepare(request, response);


        HttpSession session = request.getSession(false);
        if (session != null) {
                Object mutex = WebUtils.getSessionMutex(session);
                return handleRequestInternal(request, response);
        }

        return handleRequestInternal(request, response);
}

protected abstract ModelAndView handleRequestInternal(HttpServletRequest
request, HttpServletResponse response)
            throws Exception;
```

Memento

Defined as “Capture and restore an object's internal state”.

In a number of games, we have the feature to do an intermediate save and return to it at a later point. Implementing this needs using the Memeto pattern. We save the state of the game at the intermediate point so that we can return back to it.

Another good example is the Undo / Redo Operations in text or image editing software. Software saves the intermediate state at various points so that we can easily return back to that state.

Mediator

Defined as “Defines simplified communication between classes”.

A good example of Mediator is an Enterprise Service Bus. Instead of allowing applications to directly communicate with each other, they go through an ESB.

A good real life example is Air Traffic Controller. All the flights talk to ATC to decide the route to take. Imagine the chaos if each flight has to talk to all other flights to decide the route.

Load & Performance Testing

What are the general programming practices that are important with respect to performance in Java?

- First and Foremost - NO premature optimizations. Any optimization decision should be based on numbers or past experience. In Donald Knuth's paper "Structured Programming With GoTo Statements", he wrote: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of non critical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%."
- Minimise number of objects created:
  - Avoid String Concatenation : Use StringBuffer.
  - Avoid creating objects in Loops.
  - Consider patterns like Flyweight.
- Use correct data structures:
  - Use the right collection for a situation.
  - Use a proper domain model.

- Reduce web application overhead:
  - Small session size.
  - Use Caching where appropriate.
  - Close connections and Streams.
- Tune you database:
  - Have indexes.
  - Tune your queries.
  - If you are using hibernate, understand the internals of hibernate. Avoid N+1 Selects Problem.
  - Enable statistics on Databases.

What are the best practices in Load & Performance Testing?

Following are the best practices in terms of load and performance testing.

- Have clear performance objectives. That's the single most important objective. Decide Peak Load, Expected Response Time, Availability Required before hand.
- An application does not work on its own. It connects with a number of external interfaces. Establish clear performance expectations with the Interface Services
- The next important thing is to ensure that you mirror your production environment. A load testing environment should be the same as your

production environment. We will discuss the exact factors involved later in this article.

- Validate early : Do performance testing as early as possible.
- Make it a regular practice to use profilers in development environment. ex:JProfiler
- Make sure team is not making premature optimizations. Any optimization decision should be based on numbers or past experience. In Donald Knuth's paper "Structured Programming With GoTo Statements", he wrote: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of non critical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%."
- Have Clear Strategy on How to Handle expected load. What would be the initial settings on the application server? Do you plan to use a clustered environment? Do you need a load balancer?

What are the important factors to consider to ensure while building up the Load Test Environment?

A Load test environment should mirror production environment as much as possible:

- Application Configuration
- Application Server Configuration : Datasource properties (connections etc), JVM Memory settings, etc.
- Test Scenarios should mirror production usage. Load on different screens should mirror the usage in production.
- Ensure that the user think time is taken into consideration in the load test script.
- Consider the delays in interacting with other interfaces. If you are using stubs for interfaces, include the delay in.
- All parts of the planned production environment (like Load Balancer) should be included.
- Have same amount of data in the database as you have in production environment.

What are the important results from Profiling in Development environment?

JProfiler is a good profiling tool. The main result we expect to find from profiling is to identify the parts of the application where most request time is spent? Focus on the parts of the request which consume more than 5-10% of request time.

- Waiting for connection?
- Waiting for response from External Interface?
- Running a query on the database?
- Some loop on the application server?

Can you list some important features of JProfiler?

Important features are

- Memory profiling
- Heap Walker : See what are the objects in the Heap.
- CPU profiling : Call tree ,HotSpots - most time consuming methods list & Method statistics
- Thread profiling : Thread dumps
- Monitor profiling : all waiting and blocking situations in the JVM
- Telemetry views i.e. Graphs : Heap,Throughput, GC activity, CPU load & Database

What are the important features of Java Mission Control (Formerly JRockit Mission Control)?

- Low overhead (even in production environment)
- Captures: garbage collection pauses, memory and CPU usage, heap statistics

What are the important components in having a clear strategy to handle expected load?

- Clear Deployment Topology
- Initial Caching Strategy
- Application Server : Max Memory and Min Memory Settings - Have a clear strategy on how to play around with these?
- Database Connections - Statement Cache Size, Max Connections

What are the actions to reduce bottlenecks in an application?

Reduce demand

- Introduce Caching.
- Tuning Java Code.
- Tuning Database. (Indexing, Optimizing Queries, Optimize Hibernate settings)
- Tuning Application Server Configuration and Settings (Connection, Memory, GC etc).

Increase available resources

- Horizontal or Vertical Scaling
- More Memory
- Better CPU

Reduce slowdown due to Synchronization

- More effective collections
- More effective locking.

What are the websphere tools available for performance tuning and bottleneck analysis?

Thread and Monitor Dump Analyzer for Java

- Analyzing Java core files.
- Finds Hangs, Deadlocks, Resource contention & Bottlenecks.

Garbage Collection and Memory Visualizer

- Analyzing and visualizing verbose GC logs.
- Flag possible memory leaks, Size the Java heap correctly, Select the best garbage collection policy.

HeapAnalyzer

- Analyse Heap Dumps to find memory leaks.

PMI (Performance Monitoring Infrastructure)

- Can be switched in the websphere admin console.

- Results can be viewed in Tivoli Performance Viewer (TPV)(WAS admin console)
- Monitors JDBC Connection Pools, JVM Runtime. HeapSize, Request Count, Average time taken by servlet etc

Continuous Integration

What is Continuous Integration?

Continuous Integration can be defined as “Building software and taking it through as many tests as possible with every change”.

Why is Continuous Integration important?

Two important reasons:

- Defects found early cost less to fix : When a defect is found immediately after a developer codes it, it takes 10x times less time to fix it compared to finding the defect a month later.
- Reduced Time to Market : Software is always tested. So, it is always ready to move to further environments.

How is Continuous Integration Implemented?

Different tools for supporting Continuous Integration are Hudson, Jenkins and Bamboo. Jenkins is the most popular one currently. They provide integration with various version control systems and build tools.

What are the success factors for Continuous Integration?

Implementing the tools for Continuous Integration is the easy part. Making best use of Continuous Integration is the complex bit. Are you making the best use of your continuous integration setup? Here are the things you would need to consider.

- How often is code committed? If code is committed once a day or week, the CI setup is under utilised. Defeats the purpose of CI.
- How is a failure treated? Is immediate action taken? Does failures promote fun in the team?
- What steps are in continuous integration? More steps in continuous integration means more stability.
    - Compilation
    - Unit Tests
    - Code Quality Gates
    - Integration Tests
    - Deployment
    - Chain Tests
- More steps in continuous integration might make it take more time but results in more stable application. A trade-off needs to be made.
    - Run Steps a,b,c on a commit.

- Run Steps d & e once every 3 hours.

- How long does a Continuous Integration build run for?
    - One option to reduce time taken and ensure we have immediate feedback is to split the long running tests into a separate build which runs less often.

Security

What are the different things to consider regarding security of a web application?

Security related consideration can be split into these parts

- User Authentication and Authorization
- Web Related Issues
- External Interfaces
- Infrastructure Related Security

What are the important things to consider regarding user authentication and authorization?

Following are the important considerations:

- Proper separation of authenticated and unauthenticated resources. These can be split into separate deployable units if possible.
- Proper use of filters to ensure that the configuration for authenticated resources is centralized.
- Use a proper framework like Spring Security to implement authorization.

What are the important factors to consider when exposing an application to Internet?

OWASP (Open Web Application Security Project) is normally a great starting point. Important factors to consider are

- Validaton of user data : Ensure they are validated also in Business Layer.
- SQL Injection : Never build sql queries using string concatenation. Use a Prepared Statement. Even better, use Spring JDBCTemplate or frameworks like Hibernate, iBatis to handle communication with database.
- XSS - Cross Site Scripting : Ensure you check against a white list of input characters.
- Avoid using Old versions of software

What are important security factors to consider in communicating with external interfaces?

Security for web services (over JMS or HTTP) has to be handled at two levels : Transport level and Application level.

For HTTP based services, SSL is used to exchange certificates (HTTPS) to ensure transport level security. This ensures that the server (service producer) and client (service consumer) are mutually authenticated. It is possible to use one way SSL authentication as well.

For JMS based services, transport level security is implemented by controlling access to the Queues.

At the application level (for both JMS and HTTP based services), security is implemented by transferring encrypted information (digital signatures, for example) in the message header (SOAP Header). This helps the server to authenticate the client and be confident that the message has not been tampered with.

What are the Best Practices regarding handling security for a web application?

Best practices are:

- Threat Modelling : Do threat modelling and understand the various security threats posed to the application
- Static Security Analysis : Use a static security analysis tool like Fortify.
- Educate Developers and Testers : Most important part. Developers and Testers should be aware of the latest security threats.
- Dynamic Security Tests : Dynamic security tests done by a professional security testing team should be an important part of the release cycle. It is preferable to do this as early as possible.

OFFERS ON UDEMY COURSES

Java EE Design Patterns	SPECIAL OFFER
Spring MVC in 25 Steps	SPECIAL OFFER
JSP Servlets – 25 Steps	SPECIAL OFFER
Java Interview Guide	SPECIAL OFFER
Java OOPS Concepts	SPECIAL OFFER
Mockito – with 25 JUnit Examples	SPECIAL OFFER
Maven	SPECIAL OFFER

About in28Minutes

- At in28Minutes, we ask ourselves one question everyday. How do we help you learn effectively - that is more quickly and retain more of what you have learnt?
- We use Problem-Solution based Step-By-Step Hands-on Approach With Practical, Real World Application Examples.
- Our success on Udemy and Youtube (2 Million Views & 12K Subscribers) speaks volumes about the success of our approach.
- While our primary expertise is on Development, Design & Architecture Java & Related Frameworks (Spring, Struts, Hibernate) we are expanding into the front-end world (Bootstrap, JQuery, Angular JS).

Our Beliefs

- Best Courses are interactive and fun.
- Foundations for building high quality applications are best laid down while learning.

Our Approach

- Problem Solution based Step by Step Hands-on Learning
- Practical, Real World Application Examples.
- We use 80-20 Rule. We discuss 20% things used 80% of time in depth. We touch upon other things briefly equipping you with enough knowledge to find out more on your own.
- We will be developing a demo application in the course, which could be reused in your projects, saving hours of your effort.
- We love open source and therefore, All our code is open source too and available on Github https://github.com/in28minutes?tab=repositories.
- A preview of all our courses is on YouTube. Check out our Most Watched YouTube-Videos https://www.youtube.com/playlist?list=PLBBog2r6uMCQhZaQ9vUT5zJWXzz-f49k1.