2020

Interview Doc

FOR JAVA & AUTOMATION

SATISH NAIDU PARIMI

Table of Contents

OOP Building blocks

In Object oriented programming object contains functionality of the application

To implement an object, we implement 4 concepts which are

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Abstraction

Abstraction: is about identifying the essential details or characteristics of an object with the objective of hiding complexity of how they are implemented

In Java we achieve abstraction by using

- Interfaces
- Abstract Class

Encapsulation

Encapsulation: which is about hiding information, deciding what an object should expose to the world

Encapsulation means making members private and provide access to the outside world through getters and setters. And Yes, Encapsulation do more than this

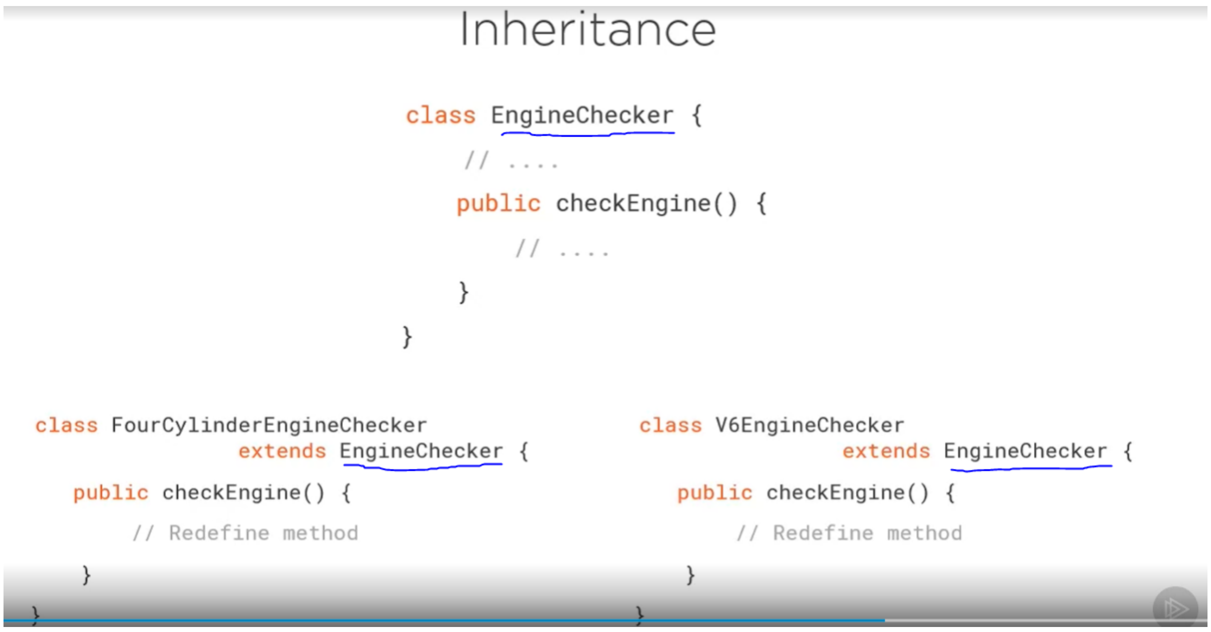
And Encapsulation separates the behavior and protect the parts of the application.

Inheritance

Inheritance: the process by one class can inherit the methods and properties from another class

Inheritance lets allow you build classes based on the other class to avoid duplicate code

In Java Inheritance achieved by extending a class or implementing an interface



Inheritance

```
class EngineChecker {
    // ....
    public checkEngine() {
        // ....
    }
}
```

```
class FourCylinderEngineChecker
                    extends EngineChecker {
    public checkEngine() {
        // Redefine method
    }
}
```

```
class V6EngineChecker
                    extends EngineChecker {
    public checkEngine() {
        // Redefine method
    }
}
```

Polymorphism:

When one class inherits from another, polymorphism allows stand in for the super class. If we need new functionality we can grab in subclass and the actual object type is decided at run time

In Addition to creating objects using the just base class Engine Checker, we can create objects for the sub class using reference of type base class, right method will be called at run time based on object creation

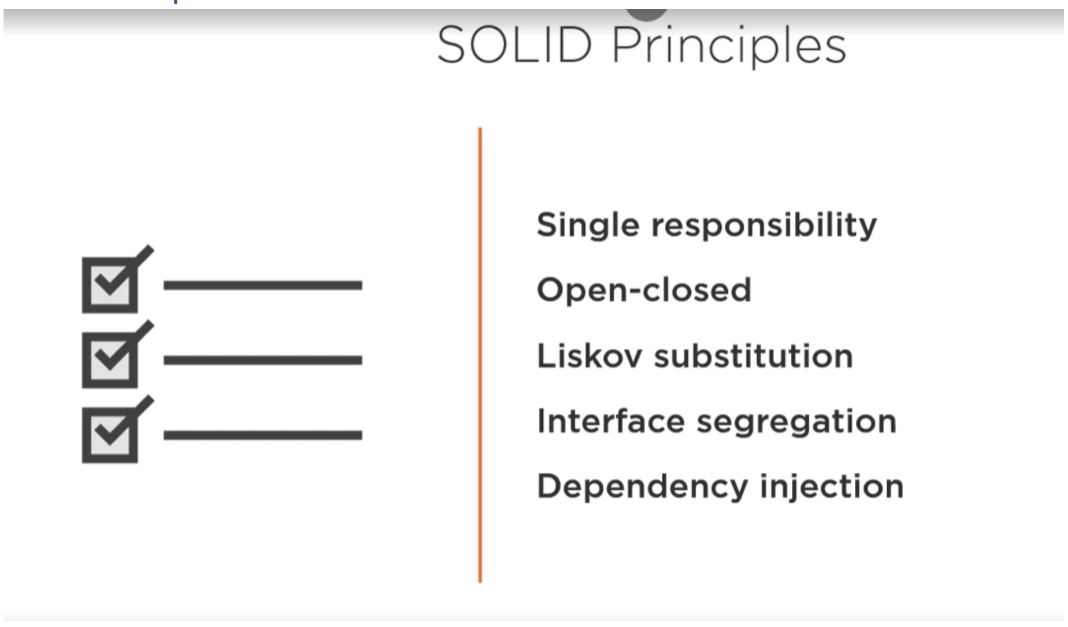
These are the building blocks of object-oriented programming

OOP Principles

Design principle are applied to the software to adopt changes easily

A well-designed software can easily change rather it breaks with small change

SOLID Principles


SOLID Principles

Single responsibility

Open-closed

Liskov substitution

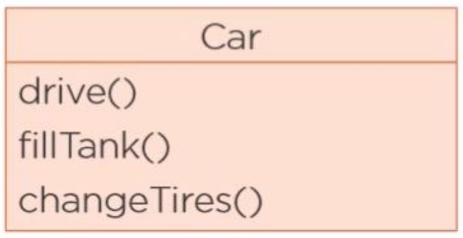
Interface segregation

Dependency injection

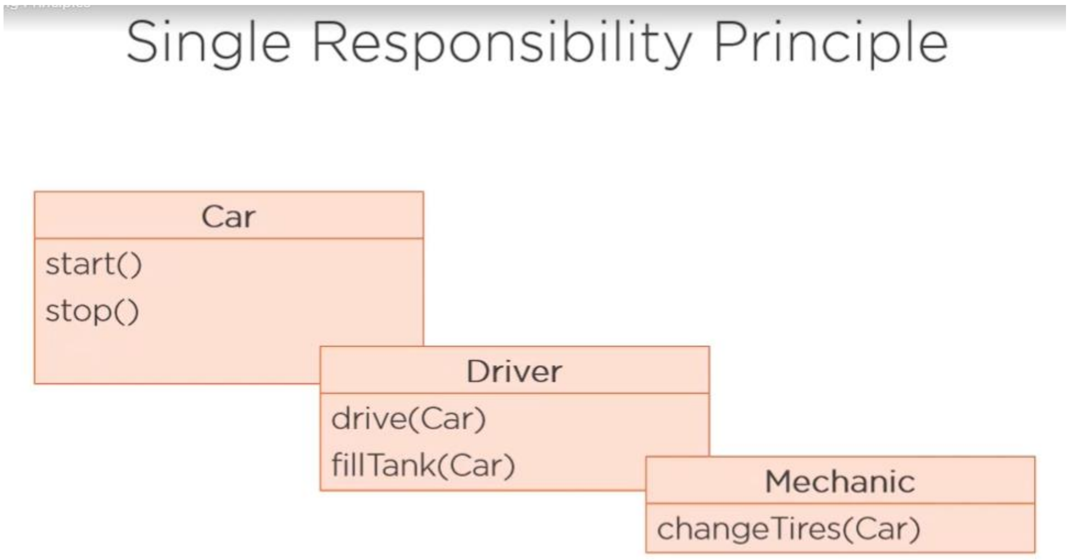
Single Responsibility

Maintain each class for one responsibility instead of many things for example

Car Class having start(), drive(), fullTank(), changeTires(), stop() sounds like this class is trying to do more things



Maybe it could be better having start() and stop() methods in the car class and move remaining to other class



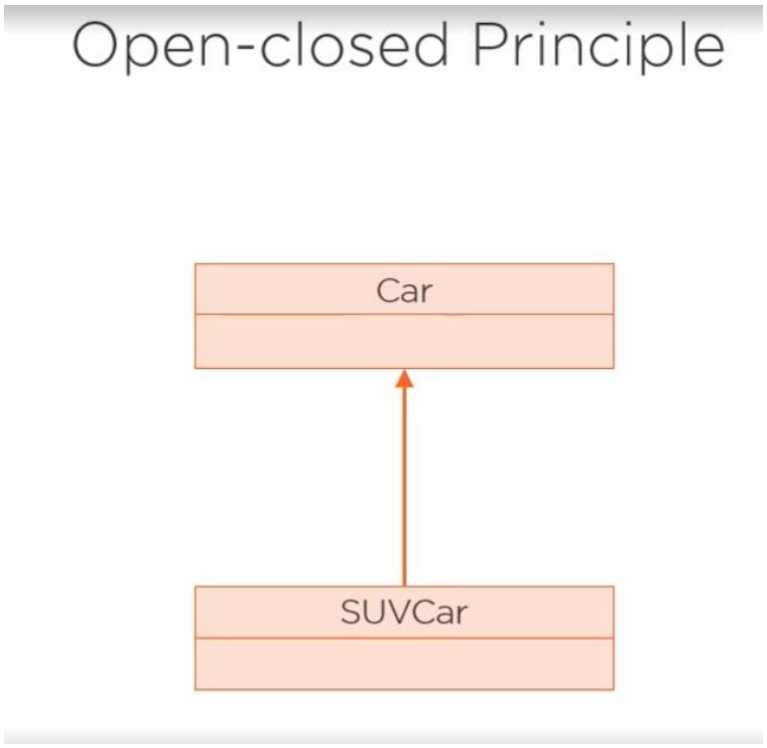
Open-Closed principle

This means open for extension and closed for modification, in the way that doesn't allow you to modify existing code

The easiest way to apply this principle is by using Inheritance

For example, to implement SUV car, we can extends Car class

Applying this principle is actually combination of Encapsulation and Abstraction



The behavior is abstracted in base class and locked for modification, when you need new or different behavior sub classes can handle the changes by extending base class

Liskov Substitution principle

Super class object should be replaced with sub class object

Same like overridden method in sub class to apply same rules for validation

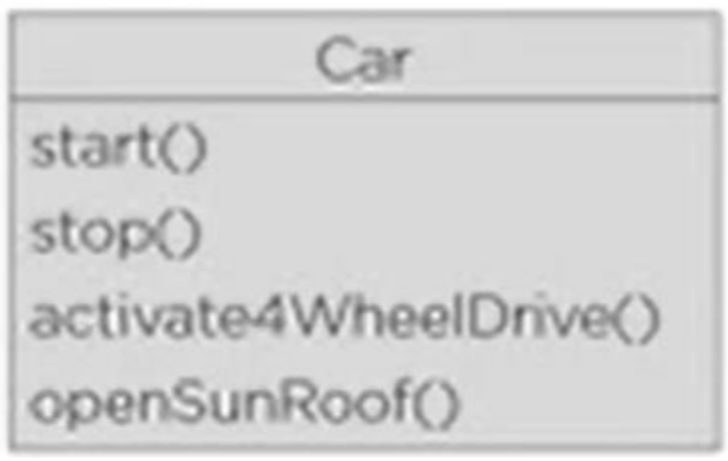
Car obj = new SuvCar() – Assginging sub class object to super class object

Interface Segregation Principle

This principle says out maintain fine-grained interfaces for specific to the client

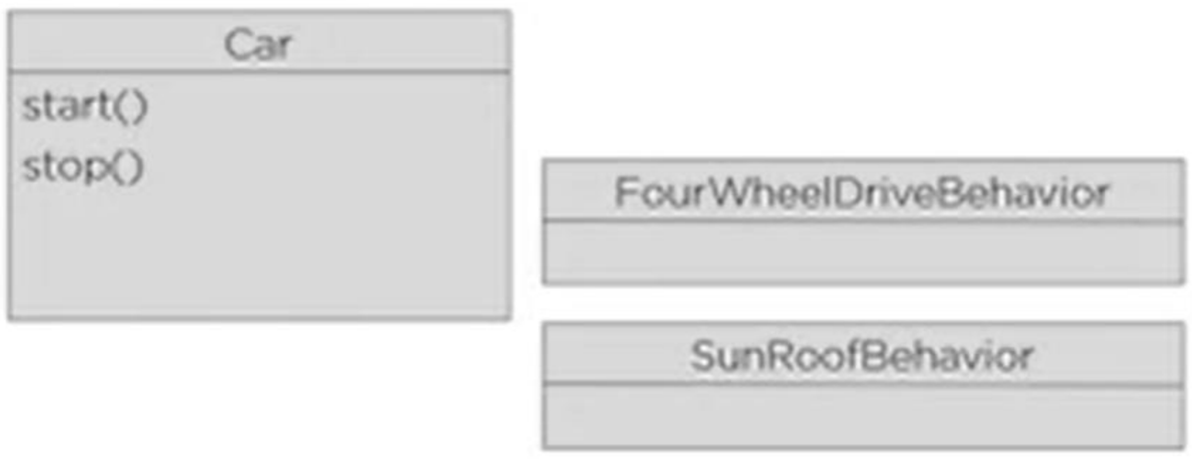
Make Interfaces as single responsibility and better to have many as modern interfaces than fewer bigger interfaces

NOTE: Segregate as many interfaces to avoid complexity



This interface having all the methods and violates the single responsibility principle

By following interface segregation principle we can break into 3 smaller interfaces based on functionality

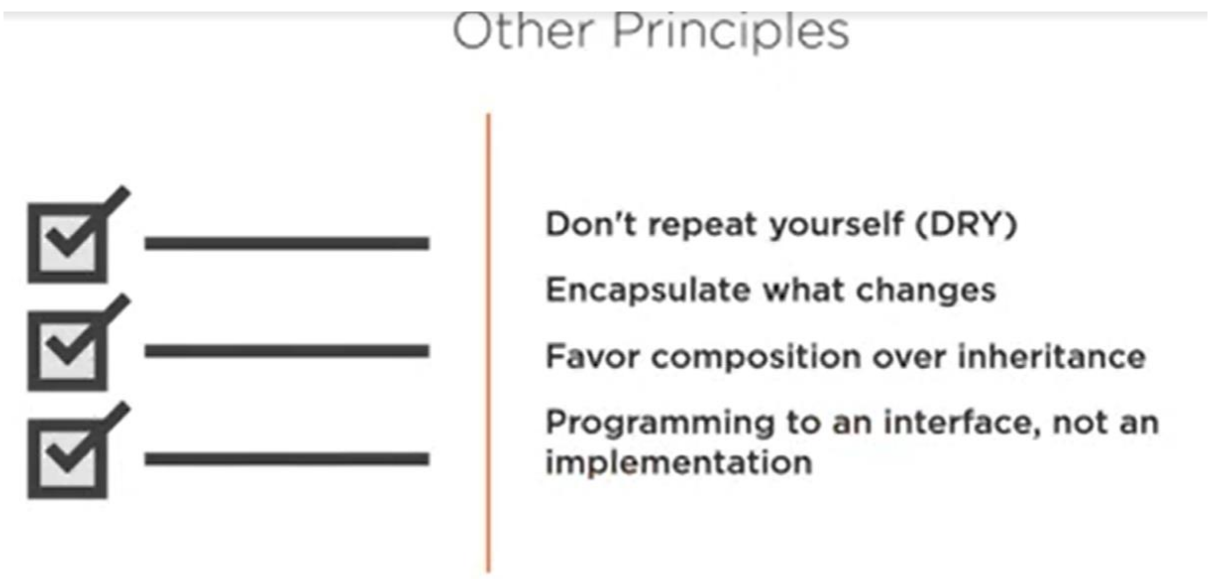


Dependency Injection:

Basically, high level modules should not depend on low level modules

The main motto of the dependency inversion is Any higher classes should always depend upon the **abstraction of the class** rather than the detail.

This aims to reduce the coupling between the classes is achieved by introducing abstraction between the layer, thus doesn't care about the real implementation.



Other Principles

Don't repeat yourself (DRY)

Encapsulate what changes

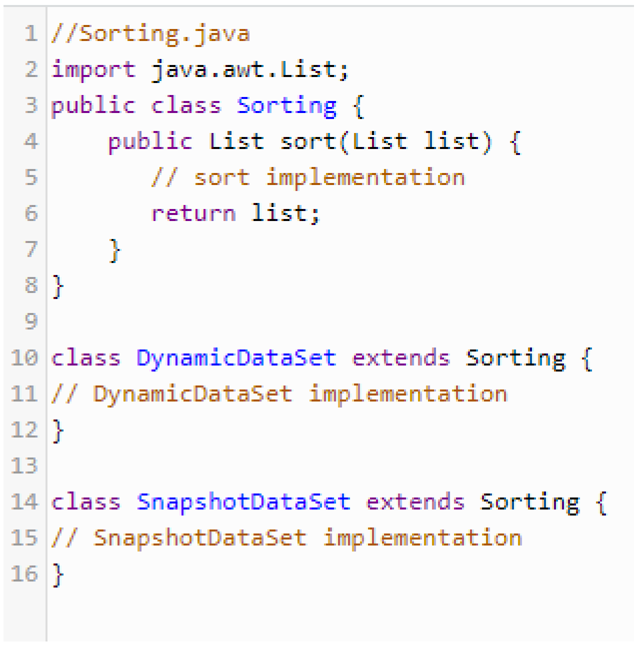
Favor composition over inheritance

Programming to an interface, not an implementation

Difference between Inheritance and Composition

Inheritance – Is a Relation ship

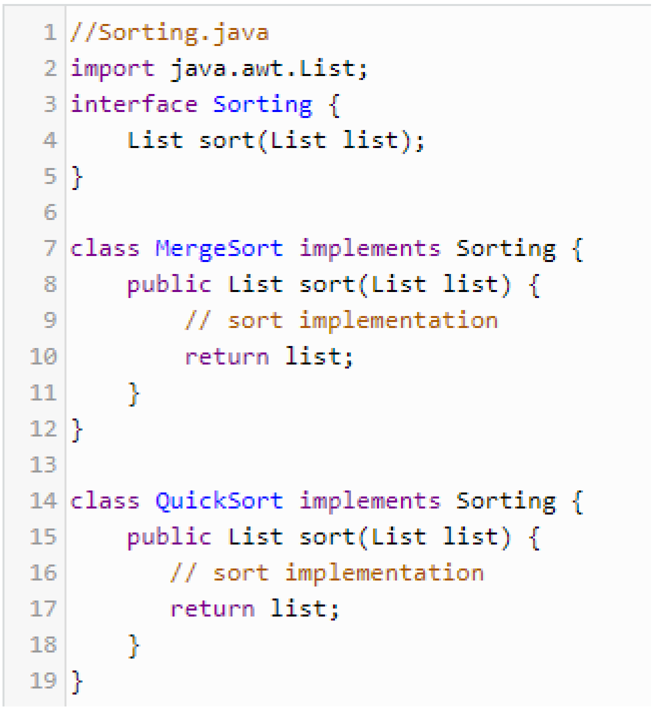
Composition – Has a Relation ship



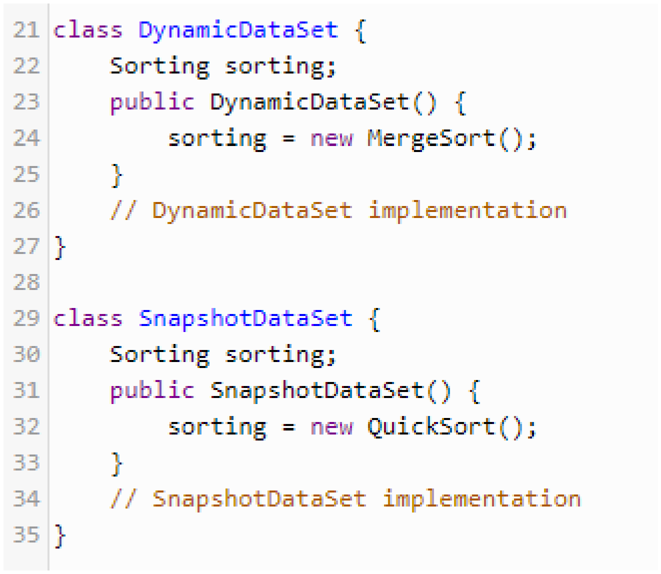
```java
//Sorting.java
import java.awt.List;
public class Sorting {
    public List sort(List list) {
        // sort implementation
        return list;
    }
}

class DynamicDataSet extends Sorting {
// DynamicDataSet implementation
}

class SnapshotDataSet extends Sorting {
// SnapshotDataSet implementation
}
```

Drawbacks in above design

- If each data set wants to implement one sorting algorithm
    a. Dynamic data set wants merge sort
    b. Snapshot Data set want quick sort
    c. Here it is major drawback to apply different sorting algorithms in “Is-A” relation ships



```java
//Sorting.java
import java.awt.List;
interface Sorting {
    List sort(List list);
}

class MergeSort implements Sorting {
    public List sort(List list) {
        // sort implementation
        return list;
    }
}

class QuickSort implements Sorting {
    public List sort(List list) {
       // sort implementation
       return list;
    }
}
```

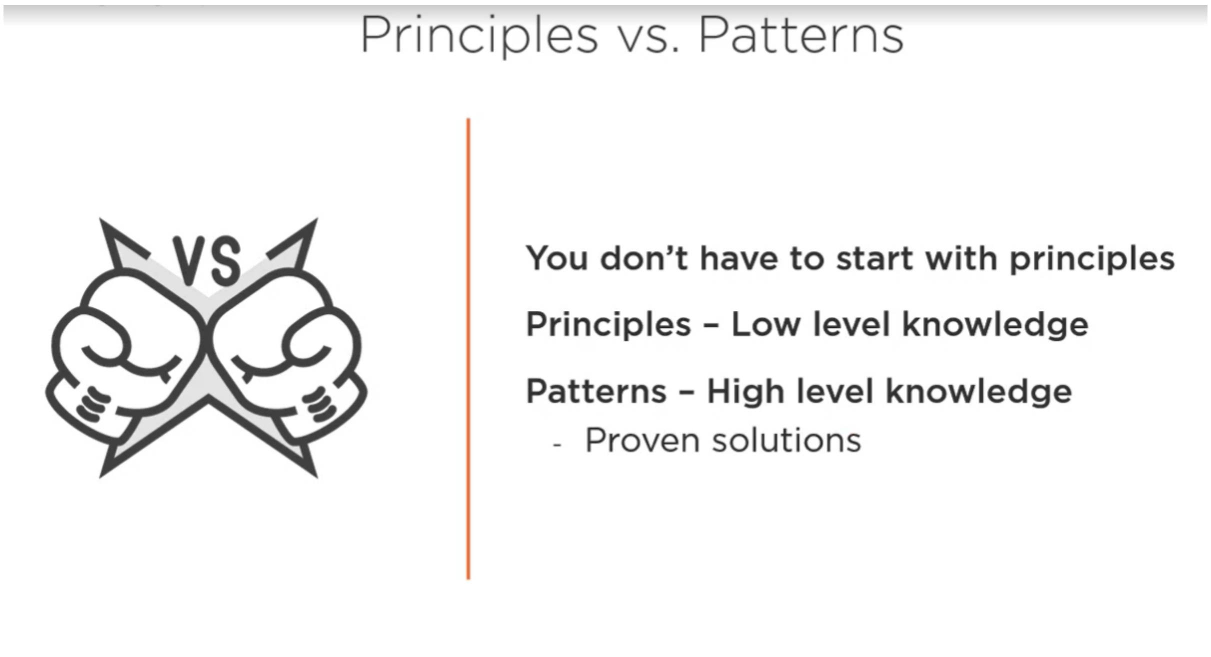


```java
class DynamicDataSet {
    Sorting sorting;
    public DynamicDataSet() {
        sorting = new MergeSort();
    }
    // DynamicDataSet implementation
}

class SnapshotDataSet {
    Sorting sorting;
    public SnapshotDataSet() {
        sorting = new QuickSort();
    }
    // SnapshotDataSet implementation
}
```

https://dzone.com/articles/composition-vs-inheritance

Principles are low level design knowledge and Patterns are High Level

Principles vs Design Patterns;

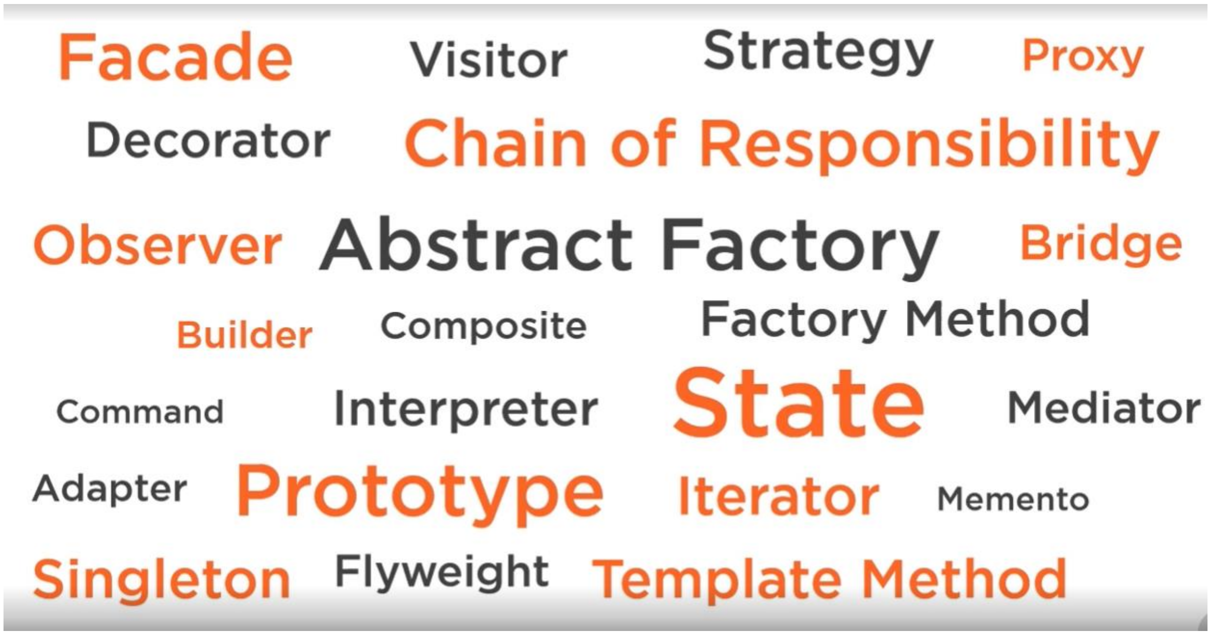

Principles vs. Patterns

You don't have to start with principles

Principles – Low level knowledge

Patterns – High level knowledge

- Proven solutions


Facade Visitor Strategy Proxy
Decorator Chain of Responsibility
Observer Abstract Factory Bridge
Builder Composite Factory Method
Command Interpreter State Mediator
Adapter Prototype Iterator Memento
Singleton Flyweight Template Method

Automation Framework

1. We developed a framework to address our product automation challenges to run tests across the clients
   - As we have product with different client interactions
   - Web Client – interacting through browsers
   - Our product is having thick clients with MS Office Com-ADD In
     - MS Excel COM Add-In
     - MS Word COM Add-In
     - MS PowerPoint COM Add-In
2. **To get automation test coverage across the clients we decided to automate at both UI and API Layers**
3. We designed the framework with the flexibility to run a single test across the layers [UI and API]
4. We thought to resolve test execution challenge from single point of contact by changing the flag.
5. We able to run a same test at UI and API layer base on the test context by using dynamic binding at component level
6. For Web automation we are using
   a. Web - **Selenium** and
   b. API Layer automaton we are using **Rest Assured Framework**
   c. For Office Client – we started using WinAppiumDriver
      i. UI Inspector like Appium Driver for mobile automation
7. Our framework is developed with maven multi module project and it is divided into 4 areas
   a. Infra
      i. Infra having all the utilities & opponents readily to build test cases at UI and API Layers
      ii. We divided Infra into 3 areas
         1. Common Module – having all the utilities for both UI, API Layers
            a. File Handling
            b. JSON Comparisons
            c. Reporting structures
            d.
         2. API Components
            a. Having all the API level components to build API tests
            b. All the operations will be handling by using Rest Assured framework
            c. We are using JSON Assert library to assert expected and actual results
         3. Selenium Components
            a. Here we develop all the selenium components to build UI browser-based test cases
            b. Here we develop selenium generic libraries and other SpreadJS utilities required for web tests automation
         4. WinAppium Driver
            a. WinAppium Driver – recently introduced framework for thick client and desktop application automation developed by Microsoft.
            b. This is an alternative of CodedUI tool

            c. In our product we implemented automation suite for Excel/Word/PPT client scenarios with WinAppium Driver framework

   b. Test Cases
      i. In this module we maintain all the test cases with complete test context information's to run tests across the layers
8. We have well established CI/CD process to run the tests across the environments
   a. As we have monthly and quarterly based releases and we need to certify the builds across the environments
      i. Initially we will validate on local Dev and QA environments on master branch
      ii. Later will move to Sandbox environment and from there we will release to operations environment later will publish on production environments
9. We implemented Automation Results dashboard with following ingredients
   a. Influx DB – time series Database to save all the test case results
   b. Grafana Server – to visualize the test results information
   c. We created Daily / weekly / monthly dashboard
10. We have specific branching strategy for automation project as well similar to our application source code
11. We follow proper code review process by creating pull requests and we have Bitbucket to maintain branches and code bases
12. We implemented Automation Results dashboard with following ingredients
   a. Influx DB – time series Database to save all the test case results
   b. Grafana Server – to visualize the test results information
   c. We created Daily / weekly / monthly dashboard

Framework Keywords

Dynamic Binding:

Test component invoking based on test type by using dynamic binding i.e. run time polymorphism

Client Code:

Consumer of components is called client code

Test Data Component

We have single source of contact for Test Data for all 3 clients (Excel, API and Web)

We use same Excel – JSON object for API Testing and Selenium test as well

Component Factory:

Resolving the diver initialization at run time based on test type

Interface Design (Interface Segregation Design Principle)

We created an Interface for each area/module , it contains module level CRUD operations

Each client can implement this interface to implement those CRUD operations

- IModel
  - ExcelModel

    - APIModel
    - SeleniumModel
- IReport
    - ExcelReport
    - APIReport
    - SeleniumReport

API Utilities in Test Pre-requisites

We use API components in test pre-requisites

For example to create a report, we need to create model, need to create model hierarchy and upload data to model as test pre-requisite and will handle all these through API in less time

This framework is implemented in efficient way to address all the challenges

Framework for Interview

We have a product with 4 clients, this product can be accessible by using

1. Web Browser
2. Office Client – XL
3. Office Client – PPT
4. Office Client Word

Earlier we have test suite with selenium test cases which are using for certifying build and it is taking 1.5 hrs for execution

In Hyderabad they extended Engg team for product development and we started building automation for regression testing and build certification

We started developing completely new framework to support all 4 clients along with API Layer.

Framework Goals:

Goal-1: Build a framework to support all 4 clients

Goal-2: Build an automation framework to run single test case across the client with flag-based mechanism

Goal-3: Single point of contact for test data management

We are using

1. **Selenium for Web application**
2. **Rest Assured Framework for API Testing**
3. **WinAppium Driver for Desktop Client automation testing**

We developed a framework such a way to integrate any component/tool easy manner

Framework Areas

1. Infra
    a. Common module
    b. API module
    c. Selenium Module
    d. Excel Module
2. Test Cases

- **Infra contains all the components for Selenium, API and XL Client**
    - We are using Interface Segregation principle to maintain components for all the clients

Component Implementation

Using Interface segregation

- Lets consider a scenario in my product , we usually build a report for financial budget and analysis based on requirement
- A report can able to build from Web Client, XL Client along with API directly

- So We created IReport – with all CRUD operation abstract methods and these methods can able implemented in child classes in respective clients
- Client code can re-use these components with respect to different clients
- IModel
    - ExcelModel
    - APIModel
    - SeleniumModel
- IReport
    - ExcelReport
    - APIReport
    - SeleniumReport

Test Case Development:

We can invoke the component and it will be resolved at run time based on test type flag

We will decide the test type by using TestNG suite parameters

We have component Factory, based on the parameter value received from TestNG suite file, we are going to initialize the Test Type at the begining of the test suite

Sample Scenario

- Before creating a report, we need to build a model and create model hierarchy like departments and need to load the data at dept level
- Once we load data we can able to summarize the data and we can prepare reports for that
- To automate this report creation, pre requisite for this test case is first we need to create a model and need to create hierarchy
- In this framework, we are using API components at test case pre -requisites and post activities

Test Case vs Component invoking:

- We are using Dynamic binding concept to invoke components from client code
- By using run time polymorphism necessary component can be invoked at run time
- We can send the test case type in test context from TestNG suite file
- Based on the parameter value it can invoke required component by initializing object

Test Data Management:

1. We have single point of contact for test data, and it can be used for all clients
2. We are maintaining test data in excel files initially while doing functional validation
3. We will capture same test data and will be used for automation as well
4. For Web Client we use to convert this Excel file - > to JSON object for test data
5. For API Testing we use to populate JSON data and validate the response

CI/CD Process

1. We Have well established CI/CD Process

Code Review Process
We have proper code review process and coding guide lines in automation projects as well similar to development project

Branching Strategy
Here we are following proper branching strategy for automation code base to maintain test cases for Quarterly based and monthly based releases

Major release feature automation will work on master branch and monthly bug fix automation will merge in both master and monthly release branches

Because master branch automation cases will fail in monthly code branch as no features are available on monthly branches

Infra contains all the components to build the test cases easily.

Reporting Dashboard with Influx DB and Grafana Server
We implemented nice Automation dashboard to check automation test suite results for multiple environments at single place

Difference Between String StringBuffer StringBuilder
String is immutable and it overrides equals() method from Object Class

String Buffer is

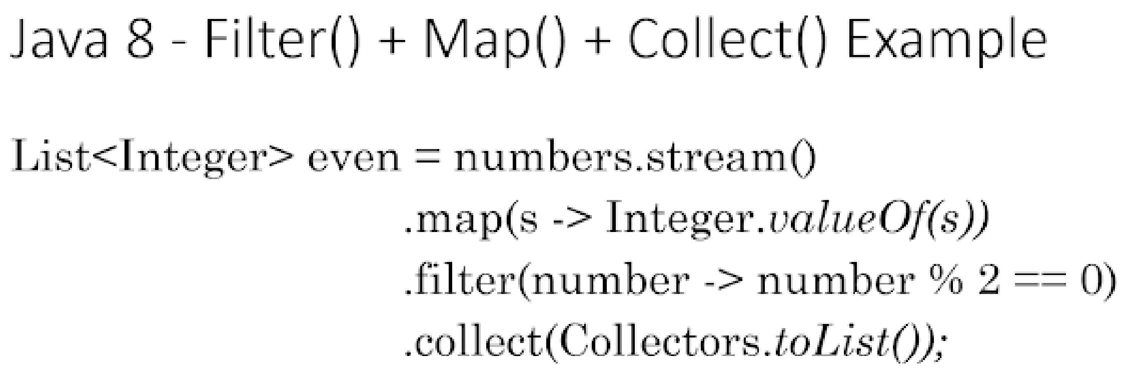
Generics In Java
We can also call it as “Type Safety” – means compiler will checks that correct values are used in correct places then there should not be any classCastException in runtime.

“Type Erasure” means the all the extra information using generics added into source code will be removed from bytecode generated from it.

Data Collection with Generics

List<Integer> customerId = new ArrayList<Integer>();

Java8 Features



Java 8 - Filter() + Map() + Collect() Example

```
List<Integer> even = numbers.stream()
                            .map(s -> Integer.valueOf(s))
                            .filter(number -> number % 2 == 0)
                            .collect(Collectors.toList());
```

map ()– to get each element from List

filter() – to execute each condition on each element received from map

reduce() -

Data Structures

Difference between Stack and Queue

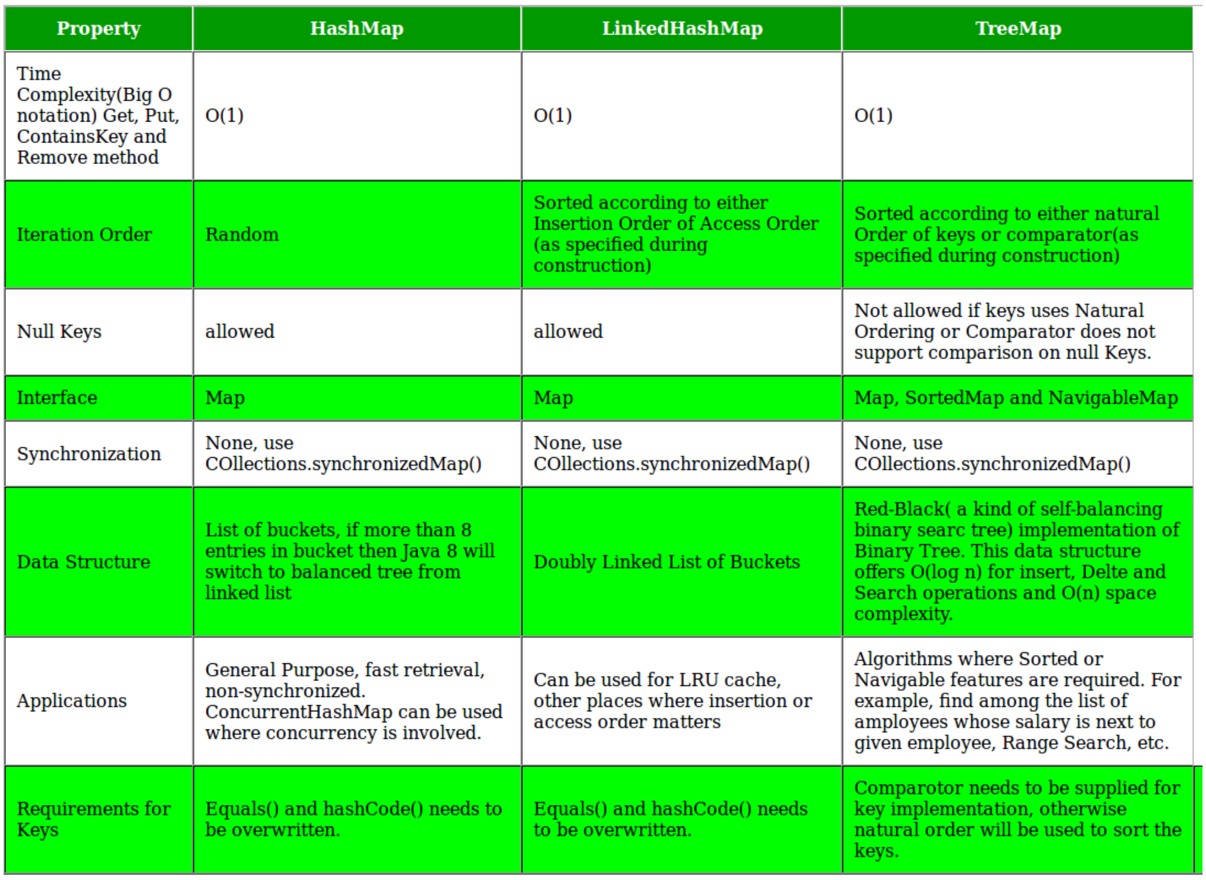
Stack: Last in First Out, can be implemented using arrays, Linked List or other forms

Queue: First in First Out [Ex: Q at bus stop, background job processing

Stack Ex:

Map comparison

https://www.techiedelight.com/difference-between-hashmap-treemap-linkedhashmap-java/



Property	HashMap	LinkedHashMap	TreeMap
Time Complexity(Big O notation) Get, Put, ContainsKey and Remove method	O(1)	O(1)	O(1)
Iteration Order	Random	Sorted according to either Insertion Order of Access Order (as specified during construction)	Sorted according to either natural Order of keys or comparator(as specified during construction)
Null Keys	allowed	allowed	Not allowed if keys uses Natural Ordering or Comparator does not support comparison on null Keys.
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()
Data Structure	List of buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black(a kind of self-balancing binary searc tree) implementation of Binary Tree. This data structure offers O(log n) for insert, Delte and Search operations and O(n) space complexity.
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of amployees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten.	Equals() and hashCode() needs to be overwritten.	Comparotor needs to be supplied for key implementation, otherwise natural order will be used to sort the keys.

Pattern Classification

Patterns are classified in two ways

- Purpose
- Scope

Types of Purpose Patterns

- Creational
- Behavioural
- Structural

Creational

There are 5 creational patterns related to object instantiation

1. Abstract Factory
2. Builder
3. Singleton
4. Factory Method
5. Prototype

Behavioural Patterns

There are 11 patterns in this category and concern with how objects interact and distribute the responsibly

1. Visitor
2. Observer
3. Command
4. Interpreter
5. Template Method
6. Strategy
7. Chain of Responsibility
8. State
9. Mediator
10. Iterator
11. Memento

Structural Patterns

There are 7 patterns in this category, this describes how classes and objects composed to create new structures or functionality

1. Decorator
2. Façade
3. Adapter
4. Bridge
5. Composite
6. Proxy
7. Flyweight

Type of Scope based patterns

There is other hand of classify patterns by a scope into

- Class
- Objects

Class Patterns describes how relationships are between classes are primarily defined via inheritance, these relationships are established at compile time

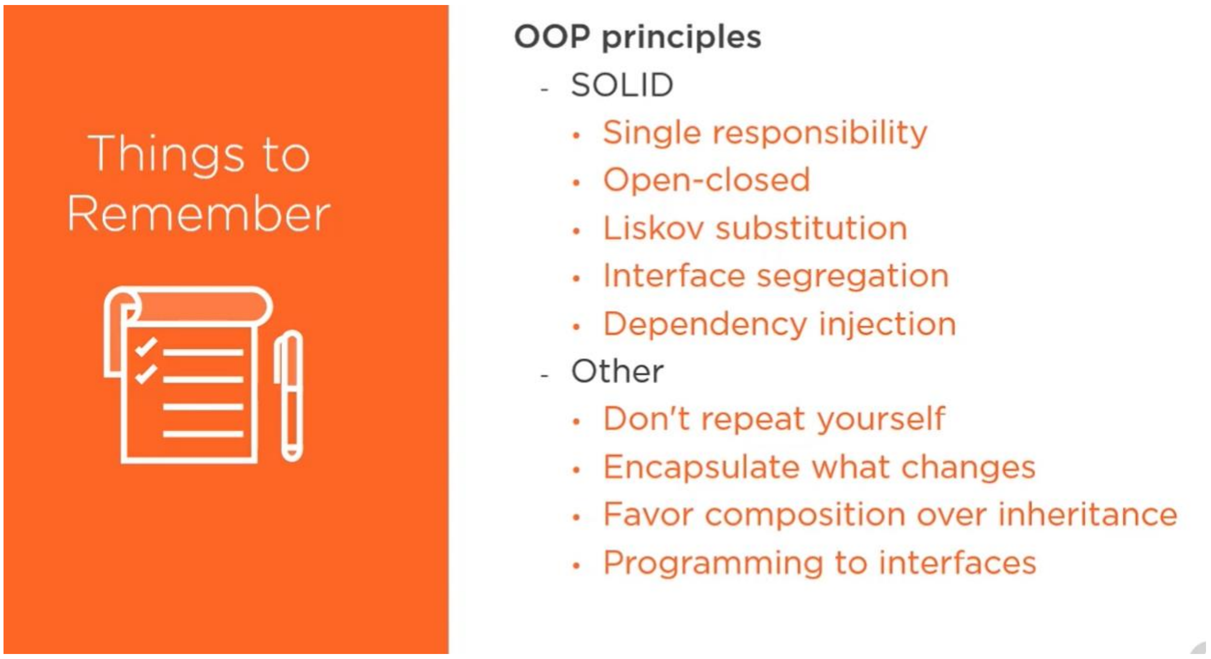
Here important thing notice that only 4 patterns in this category

1. Interpreter
2. Adapter
3. Factory Method
4. Template Method

Object Patterns

Object patterns describes relationship between objects that are primarily defined by composition, these relationships are created at run time and more dynamic and flexible than ones defined by inheritance

Design Principle Review

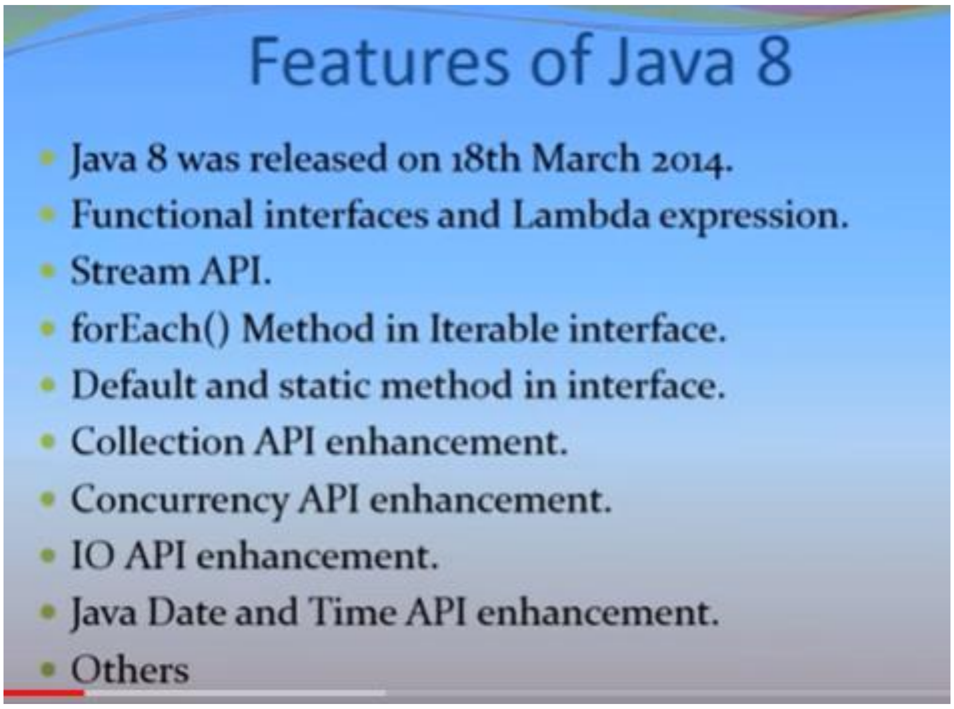

Things to Remember

OOP principles

- SOLID
  - Single responsibility
  - Open-closed
  - Liskov substitution
  - Interface segregation
  - Dependency injection
- Other
  - Don't repeat yourself
  - Encapsulate what changes
  - Favor composition over inheritance
  - Programming to interfaces

Java 8 Features

Below are the hig- level Java 8 features


Features of Java 8

- Java 8 was released on 18th March 2014.
- Functional interfaces and Lambda expression.
- Stream API.
- forEach() Method in Iterable interface.
- Default and static method in interface.
- Collection API enhancement.
- Concurrency API enhancement.
- IO API enhancement.
- Java Date and Time API enhancement.
- Others

Functional Interfaces:

1. An interface having one abstract method and any number of default and static methods
2. Functional interface also called as “Single Abstract Method Interface” or SAM Interface
3. A Method from object class can also be declared like **toString()** method

Implementation

1. @FunctionalInterface annotation is required to mark an interface as **Functional Interface**
2. This annotation will not allow to add more than one abstract method work like a guardrail

Lambda Expression:

1. Lambda expression are used to **provide the implementation of functional interfaces**
2. Lambda expression is like a function
3. .class file will not be created for them
4. As we all know Java is object oriented programming, In Java 8 they introduced Lambda expressions as Functional programming

Stream API

1. Java.util.stream API added in java 8, which has methods
   a. Filter
   b. Map ()
   c. Reduce ()
   d. Sorted ()
   e. forEach ()
2. Stream API Can work with source as
   a. Collection
   b. IO Operation
   c. Arrays
3. Operation performed in Stream doesn’t modify the original source
4. Majority of Stream API method having arguments as Functional Interfaces, so Lambda Expressions has very vast scope here

Time API

1. Before Java 8 Date class present util class and SQL class also
2. Earlier all Date classes are mutable, as they don’t provide thread safety
3. In Java 8 with Time API, they provided thread safety and Internationalization
4. Time API provides thread safety and Internationalization to work with different time zones

Other Method in Java 8

1. Apart from forEach() and Stream API we have other methods also added in collection API
   a. forEachRemaining ()
   b. removeelf()
   c. spliterator()
   d. replaceAll()
   e. compute()
   f. merge()
2. few new methods like
   a. compute()
   b. forEach()
   c. forEachEntry()

    d. forEachKey()
    e. forEachValue()
    f. merge()
    g. reduce()
    h. search()

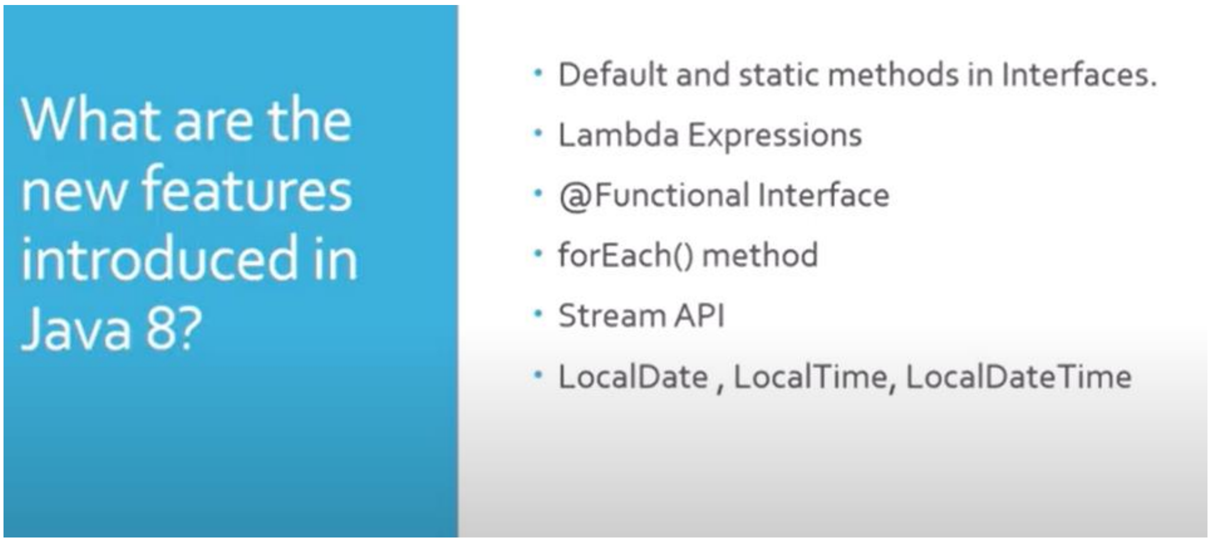
3. File new APIs
    a. Files.list(Path dirPath)
    b. Files.lines(Path path)
    c. File.find()
    d. BufferReader.lines() are added in IO package

Other Features in Java 8

1. **Optional Class:** Optional class is added java.util package to work with null pointer exception
2. **Natural and reverse Order**: Many default and static methods are added comparator interface for natural and reverse ordering of elements
3. New methods like **min()**, **max()**and **sum()** methods are added in
    a. Integer
    b. Long and
    c. Double wrapper classes
4. logicalAnd(), logicalOr(), logicalXor() methods in Boolean class
5. Many utility methods in Math class

Java8 Interview Questions

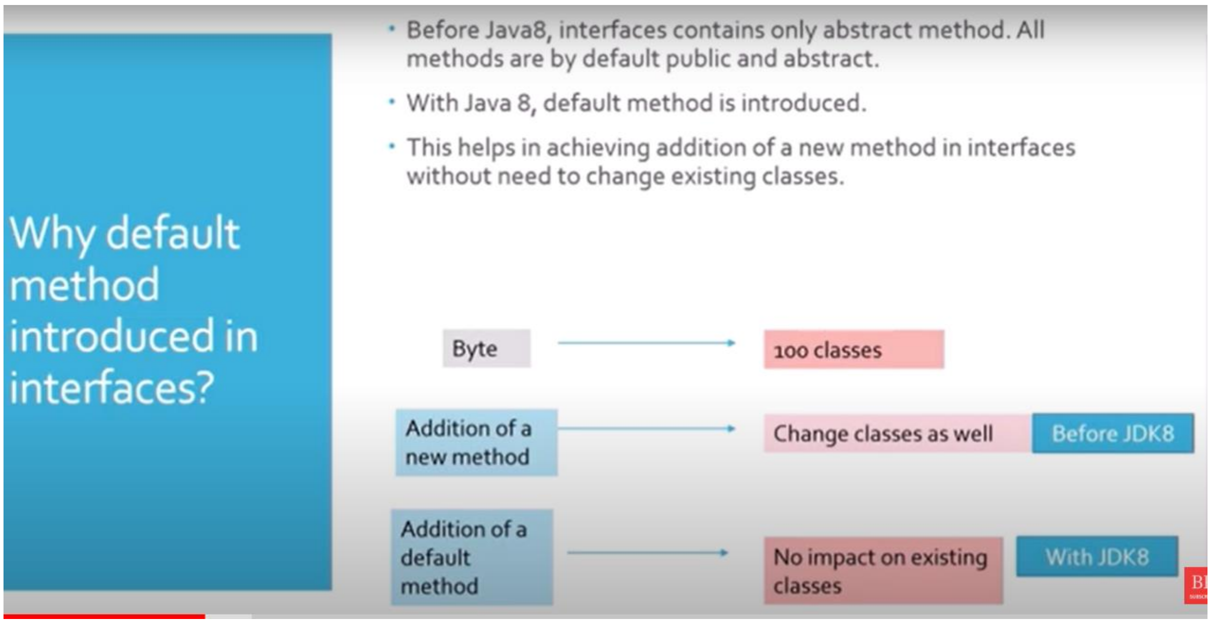
What are the new features in Java8?



What are the new features introduced in Java 8?

- Default and static methods in Interfaces.
- Lambda Expressions
- @Functional Interface
- forEach() method
- Stream API
- LocalDate , LocalTime, LocalDateTime

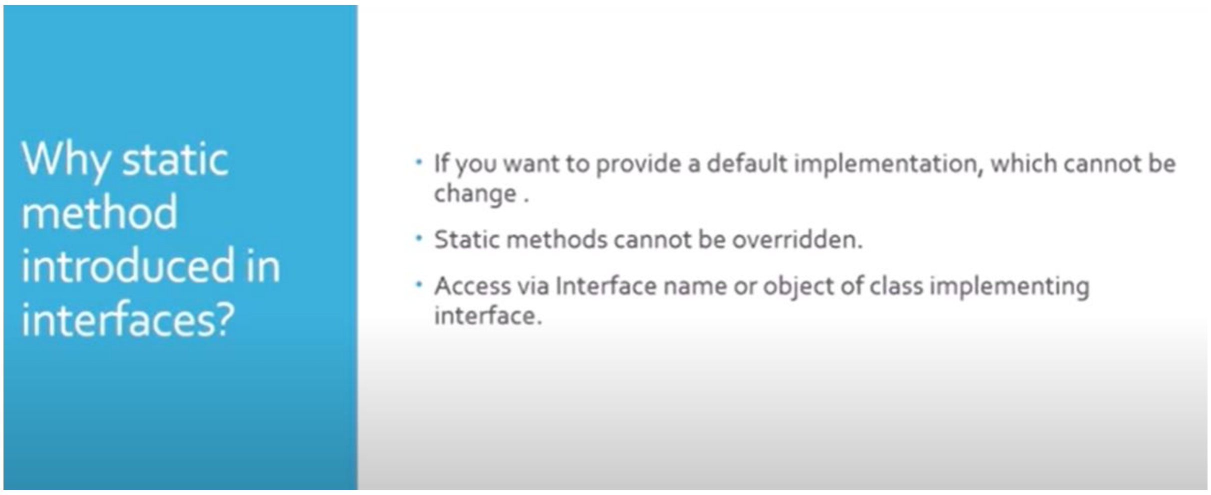
Why default method introduced in Interfaces?



Why default method introduced in interfaces?

- Before Java8, interfaces contains only abstract method. All methods are by default public and abstract.
- With Java 8, default method is introduced.
- This helps in achieving addition of a new method in interfaces without need to change existing classes.

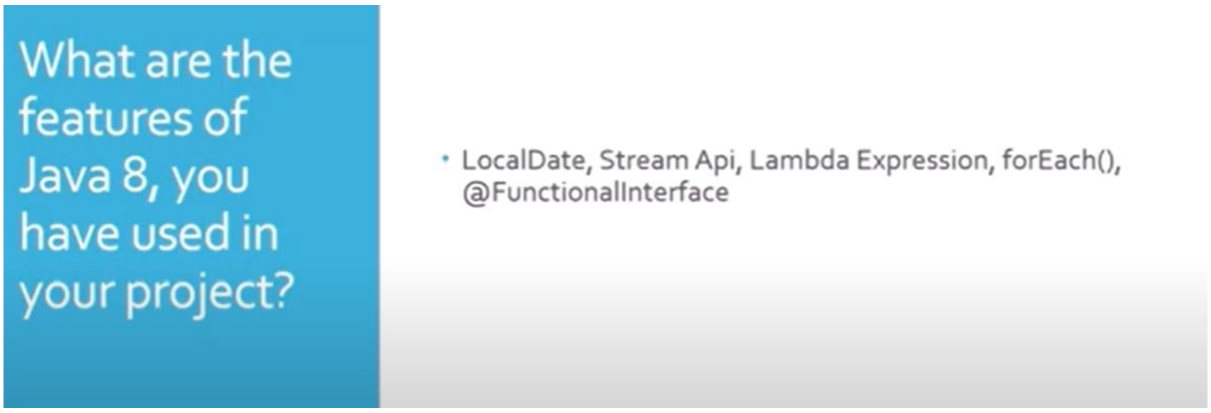
Why static methods introduced in Interfaces?



Why static method introduced in interfaces?

- If you want to provide a default implementation, which cannot be change .
- Static methods cannot be overridden.
- Access via Interface name or object of class implementing interface.

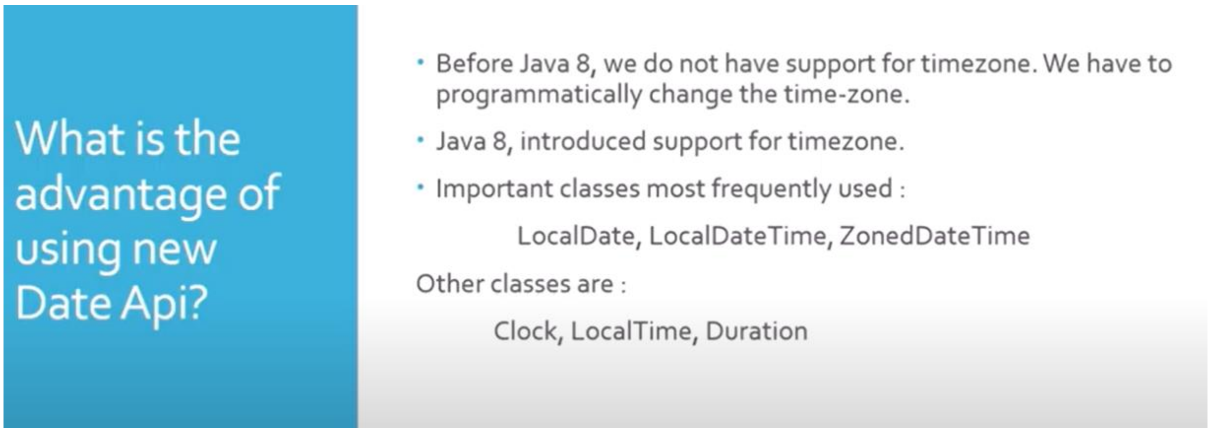
What are the features of Java8, you've used in your project



What are the features of Java 8, you have used in your project?

- LocalDate, Stream Api, Lambda Expression, forEach(), @FunctionalInterface

What are the advantages of using new Date API?



What is the advantage of using new Date Api?

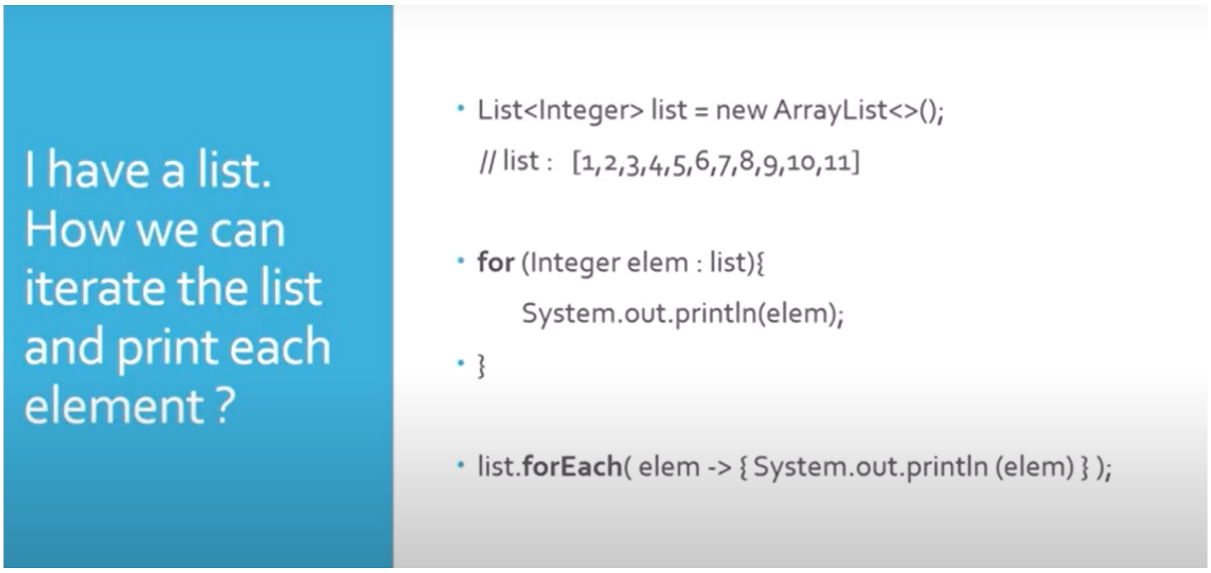
- Before Java 8, we do not have support for timezone. We have to programmatically change the time-zone.
- Java 8, introduced support for timezone.
- Important classes most frequently used :

  LocalDate, LocalDateTime, ZonedDateTime

Other classes are :

  Clock, LocalTime, Duration

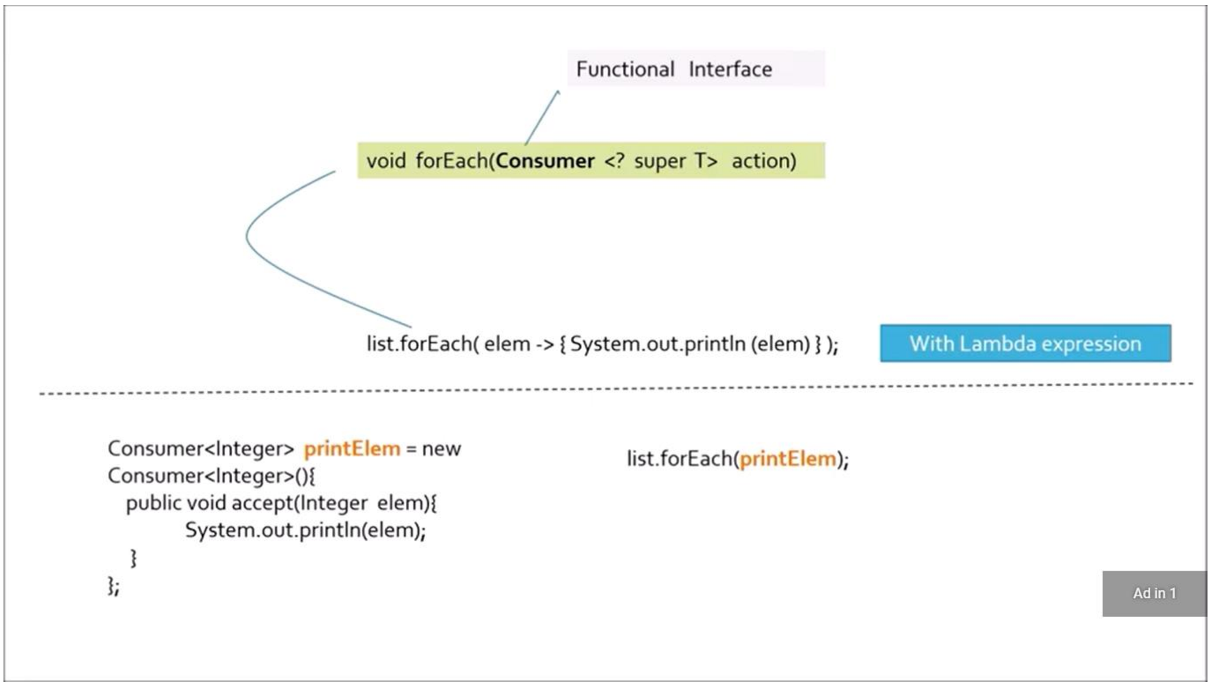
I Have a list, how we can iterate the list and print each element?



I have a list. How we can iterate the list and print each element ?

```
• List<Integer> list = new ArrayList<>();
  // list :  [1,2,3,4,5,6,7,8,9,10,11]

• for (Integer elem : list){
      System.out.println(elem);
• }

• list.forEach( elem -> { System.out.println (elem) } );
```

How forEach() is implemented using **Consumer @Functional Interface (also called as SAM Interface)**

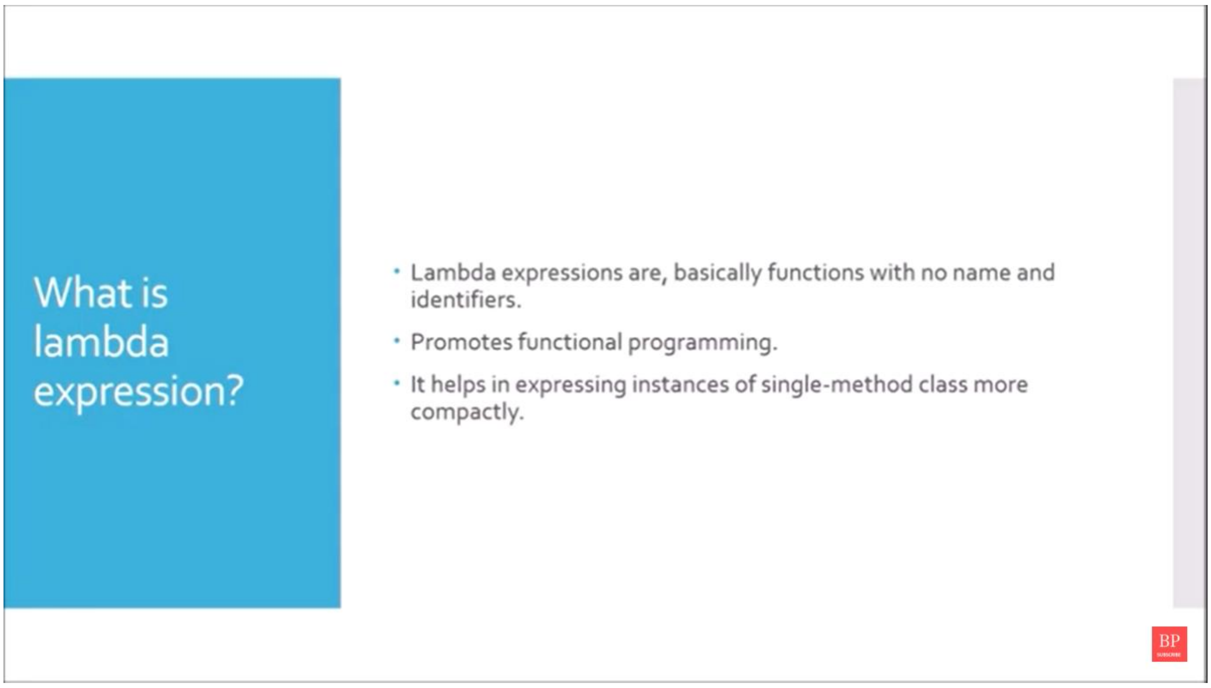


```
void forEach(Consumer <? super T>  action)

list.forEach( elem -> { System.out.println (elem) } );
```

```
Consumer<Integer>  printElem = new
Consumer<Integer>(){
   public void accept(Integer  elem){
            System.out.println(elem);
   }
};
```

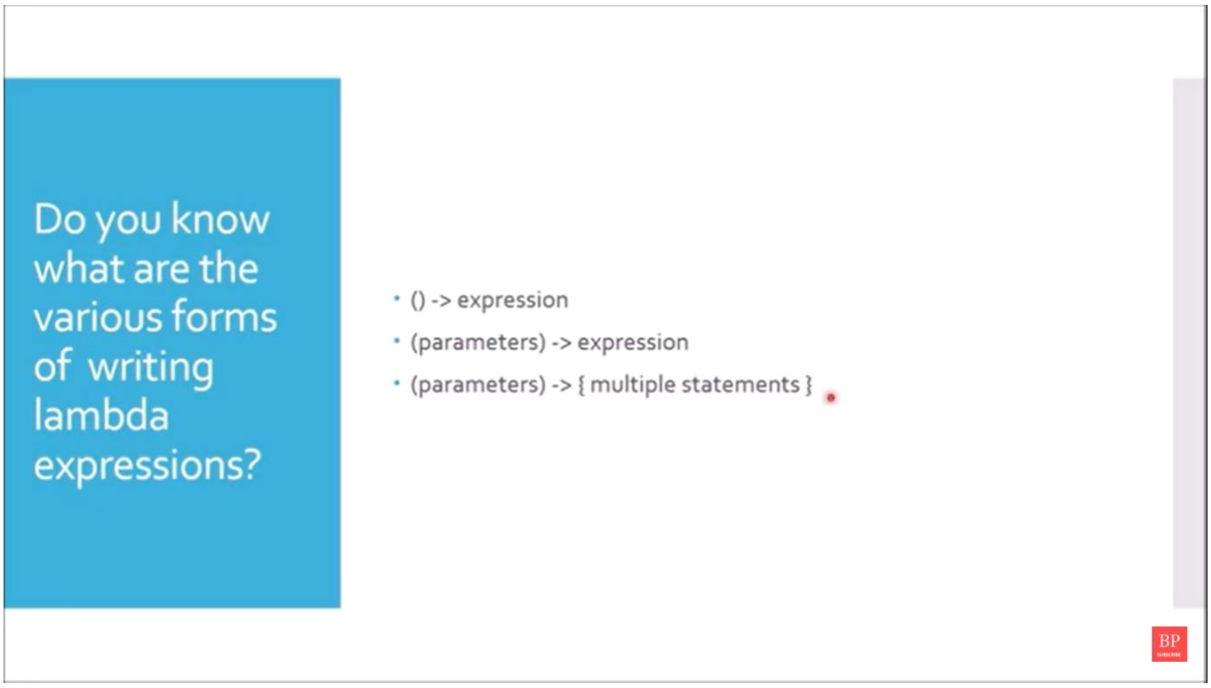
```
list.forEach(printElem);
```

What is Lambda Expression?


What is lambda expression?

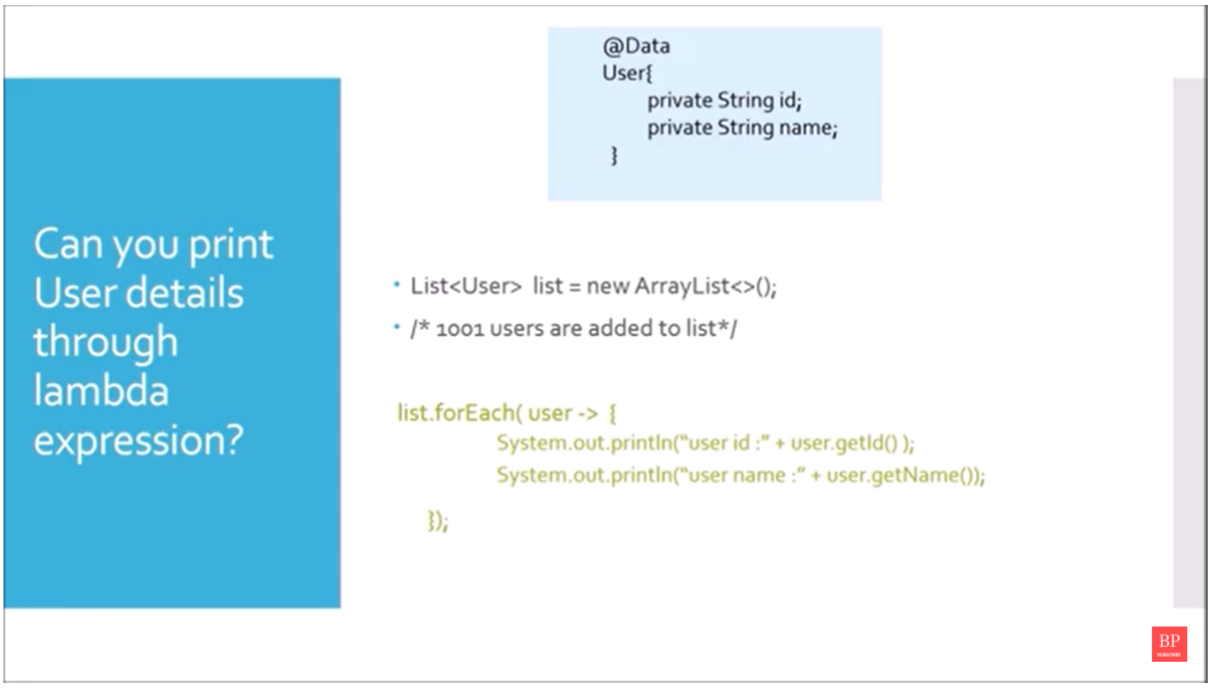
- Lambda expressions are, basically functions with no name and identifiers.
- Promotes functional programming.
- It helps in expressing instances of single-method class more compactly.

Do you know what are the various forms of writing Lambda Expressions?


Do you know what are the various forms of writing lambda expressions?

- () -> expression
- (parameters) -> expression
- (parameters) -> { multiple statements }

Can you print User Details through Lambda Expression?



Can you print User details through lambda expression?

```
@Data
User{
        private String id;
        private String name;
  }
```

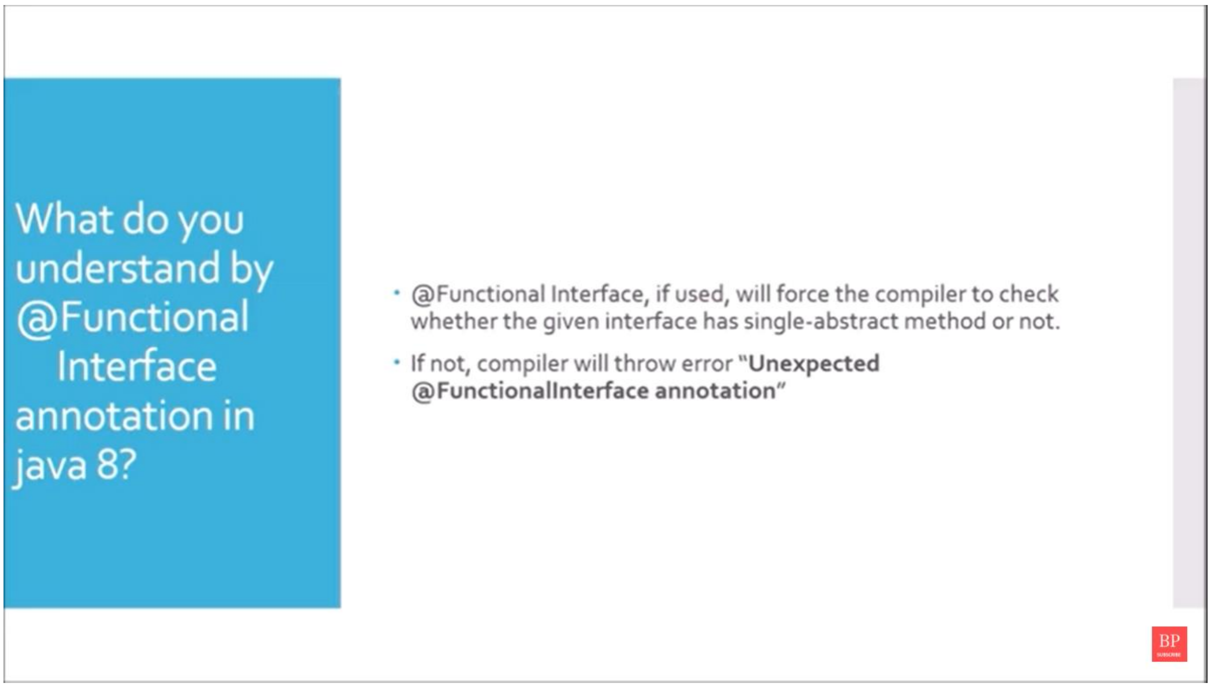
- List<User> list = new ArrayList<>();
- /* 1001 users are added to list*/

```
list.forEach( user -> {
            System.out.println("user id :" + user.getId() );
            System.out.println("user name :" + user.getName());

    });
```

BP

Refer our **com.host.java8.LambdaExpression** package examples

What do you understand by @FunctionalInterface annotation in Java8?



What do you understand by @Functional Interface annotation in java 8?

- @Functional Interface, if used, will force the compiler to check whether the given interface has single-abstract method or not.
- If not, compiler will throw error "**Unexpected @FunctionalInterface annotation**"

BP

- Functional Interface concept is available before Java 8 also, those are
  - Comparator – Functional Interface
  - Comparable – Functional Interface
  - Runnable - Functional Interface
- In Java 8 they introduced
  - @FunctionalInterface Annotation
  - Consumer – Functional Interface

How do you create a custom annotation in java?

We have two types of annotations based on creation

1. Default annotations
2. Custom annotations

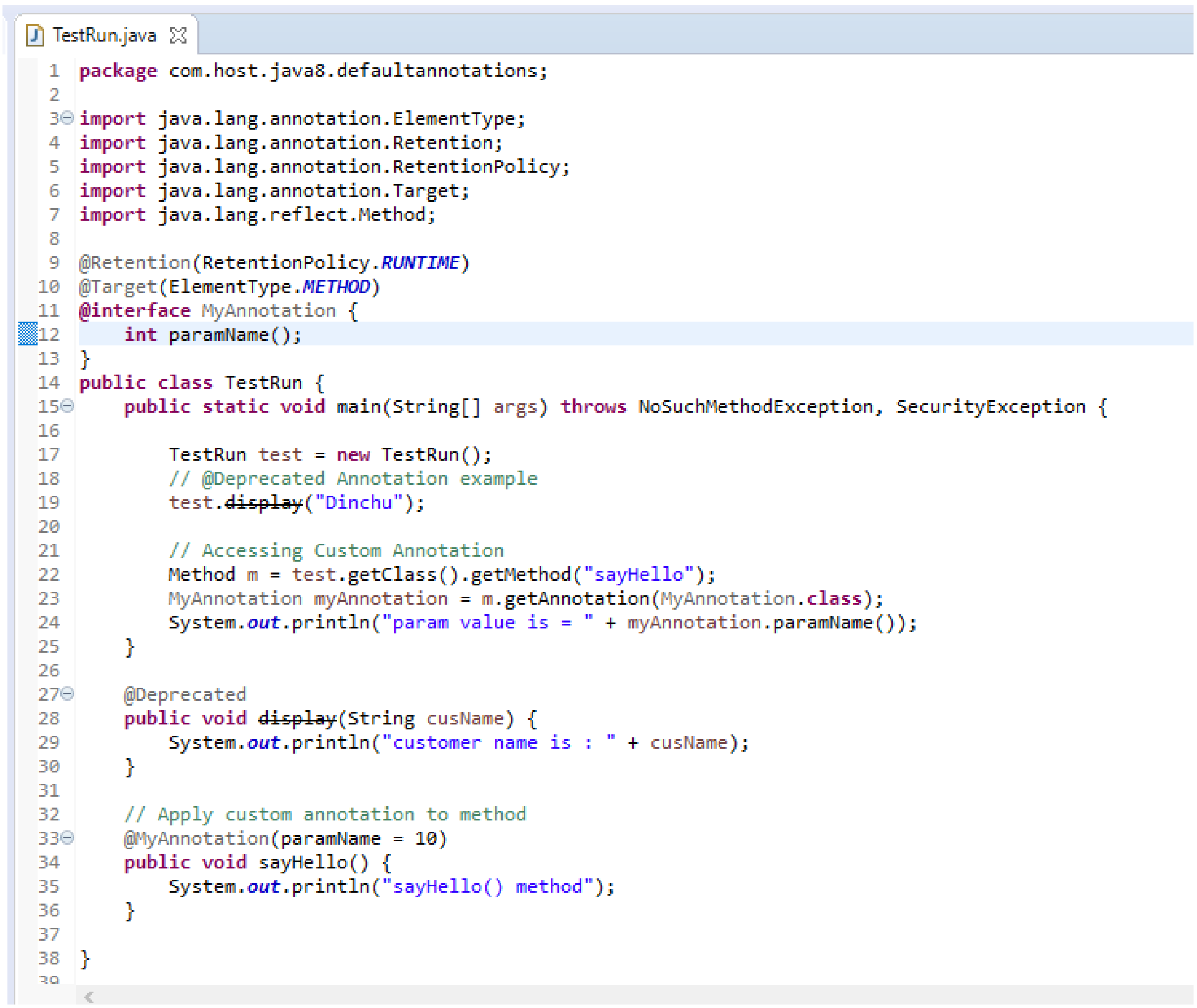
We can categorize annotations based on …

1. Marker annotations
2. Single Value Annotations
3. Multi Value Annotations

Marker Annotation: it has no methods

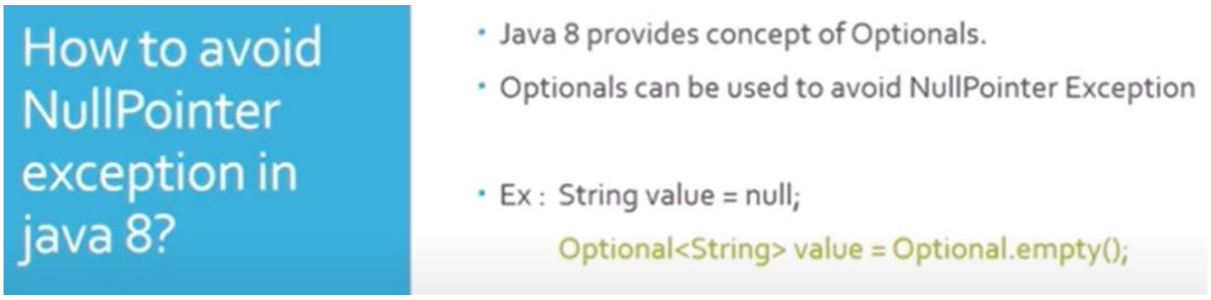
Single Value Annotation: It has one method and we can also provide default value also

Multi Value Annotation: An annotation has more than one method, It is called multi-value annotation



```java
package com.host.java8.defaultannotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.reflect.Method;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface MyAnnotation {
    int paramName();
}
public class TestRun {
    public static void main(String[] args) throws NoSuchMethodException, SecurityException {

        TestRun test = new TestRun();
        // @Deprecated Annotation example
        test.~~display~~("Dinchu");

        // Accessing Custom Annotation
        Method m = test.getClass().getMethod("sayHello");
        MyAnnotation myAnnotation = m.getAnnotation(MyAnnotation.class);
        System.out.println("param value is = " + myAnnotation.paramName());
    }

    @Deprecated
    public void ~~display~~(String cusName) {
        System.out.println("customer name is : " + cusName);
    }

    // Apply custom annotation to method
    @MyAnnotation(paramName = 10)
    public void sayHello() {
        System.out.println("sayHello() method");
    }

}
```

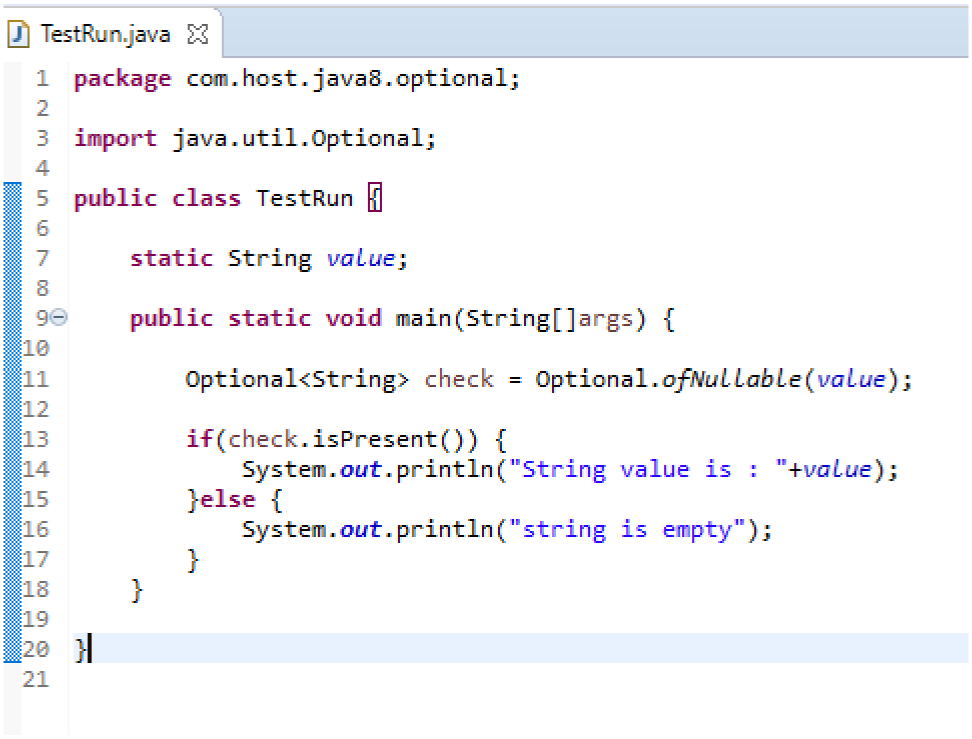
How to avoid null pointer exception in java 8?



How to avoid NullPointer exception in java 8?

- Java 8 provides concept of Optionals.
- Optionals can be used to avoid NullPointer Exception
- Ex : String value = null;

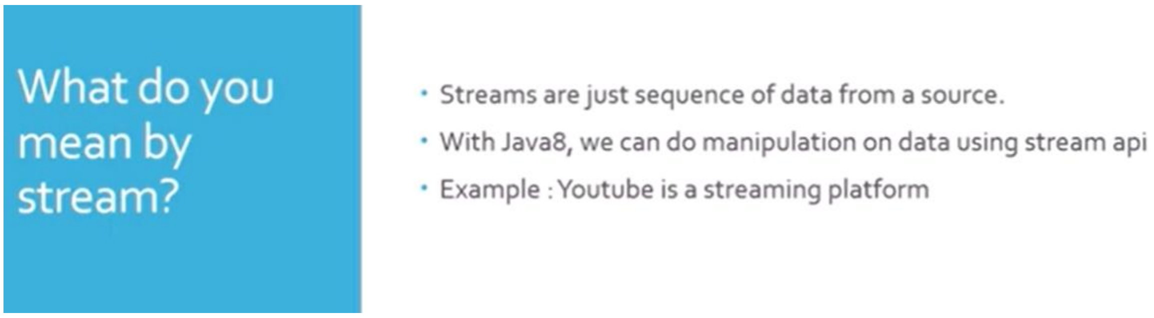
  Optional<String> value = Optional.empty();



TestRun.java

```java
package com.host.java8.optional;

import java.util.Optional;

public class TestRun {

    static String value;

    public static void main(String[]args) {

        Optional<String> check = Optional.ofNullable(value);

        if(check.isPresent()) {
            System.out.println("String value is : "+value);
        }else {
            System.out.println("string is empty");
        }
    }

}
```

What do you mean by Stream?

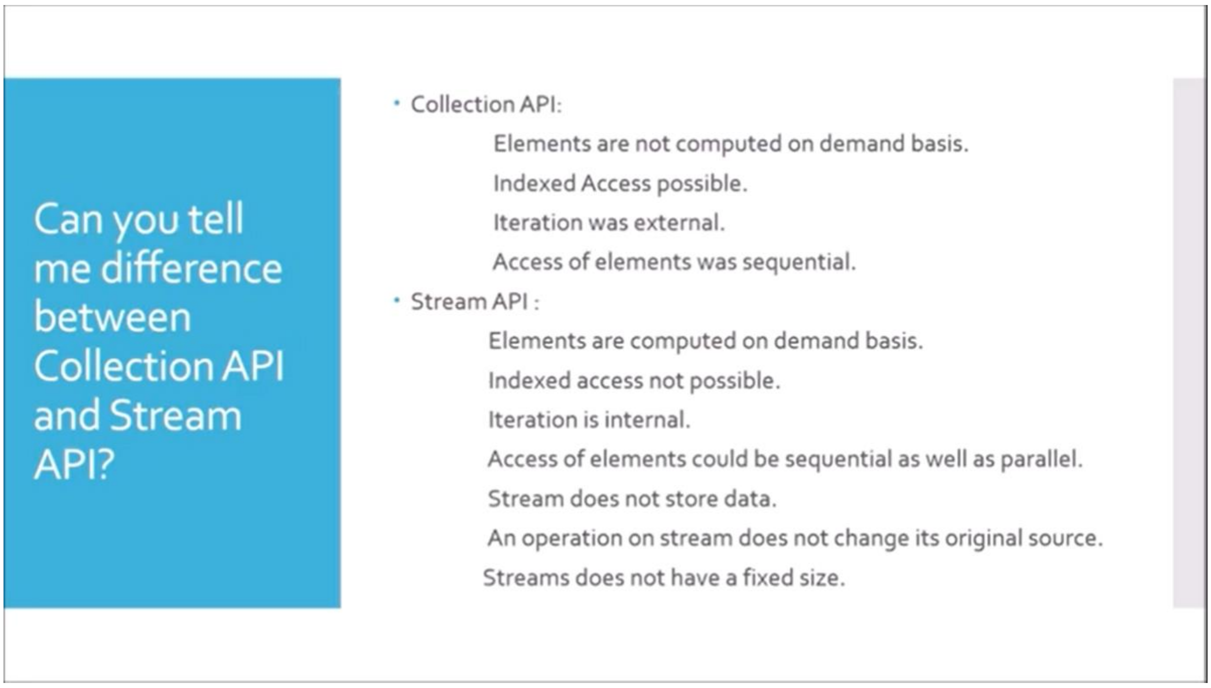


What do you mean by stream?

- Streams are just sequence of data from a source.
- With Java8, we can do manipulation on data using stream api
- Example : Youtube is a streaming platform

To manipulate the data receiving from source like collection and array

Can you tell me difference between Collection API and Stream API?


Can you tell me difference between Collection API and Stream API?

- Collection API:
  - Elements are not computed on demand basis.
  - Indexed Access possible.
  - Iteration was external.
  - Access of elements was sequential.
- Stream API :
  - Elements are computed on demand basis.
  - Indexed access not possible.
  - Iteration is internal.
  - Access of elements could be sequential as well as parallel.
  - Stream does not store data.
  - An operation on stream does not change its original source.
  - Streams does not have a fixed size.

What are the functional interfaces you have used in your project, introduced with java 8?

Following are the Functional Interfaces introduced with Java8

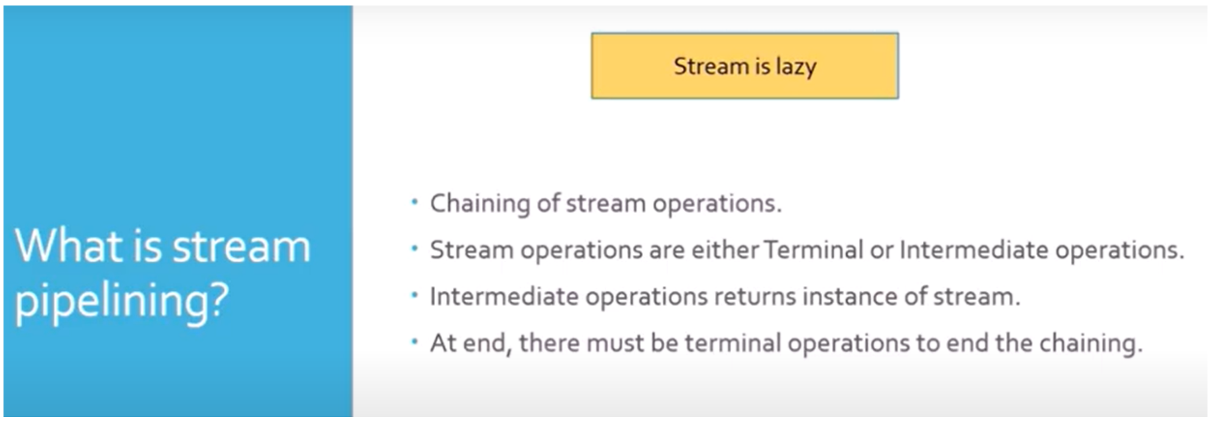
- Function
- Consumer
- Supplier
- Predicate

Function: Takes 1 argument and return result

Predicate: Takes 1 argument and return Boolean

All these functional interfaces present in java.util.function package

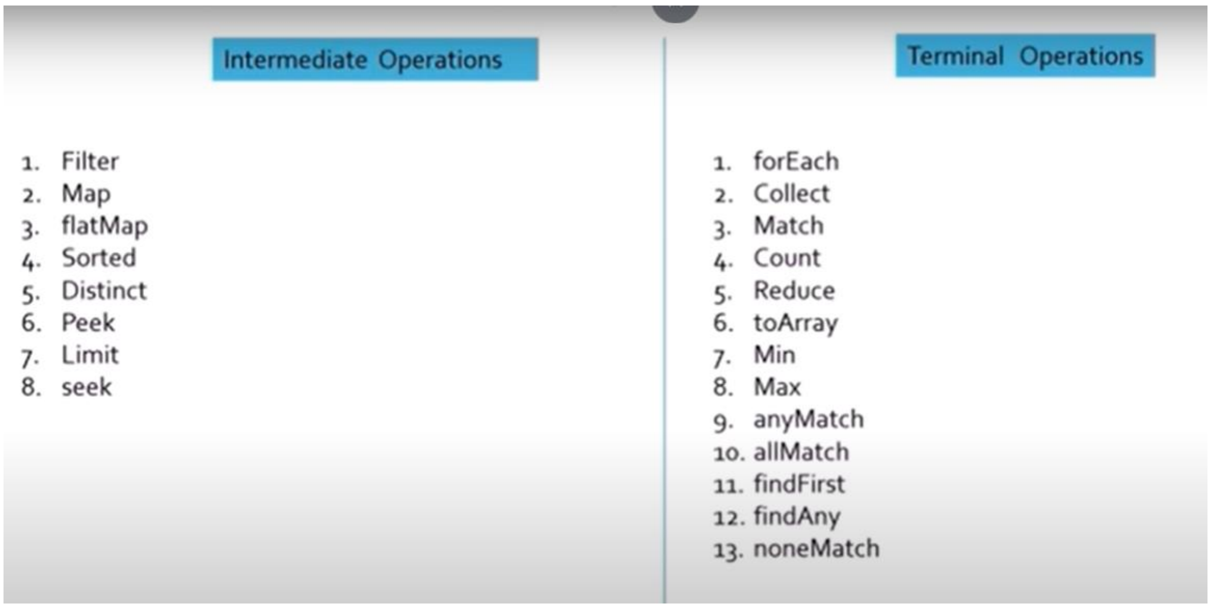
What is Stream pipelining?


What is stream pipelining?

Stream is lazy

- Chaining of stream operations.
- Stream operations are either Terminal or Intermediate operations.
- Intermediate operations returns instance of stream.
- At end, there must be terminal operations to end the chaining.

Intermediate and Terminal Operations



Intermediate Operations

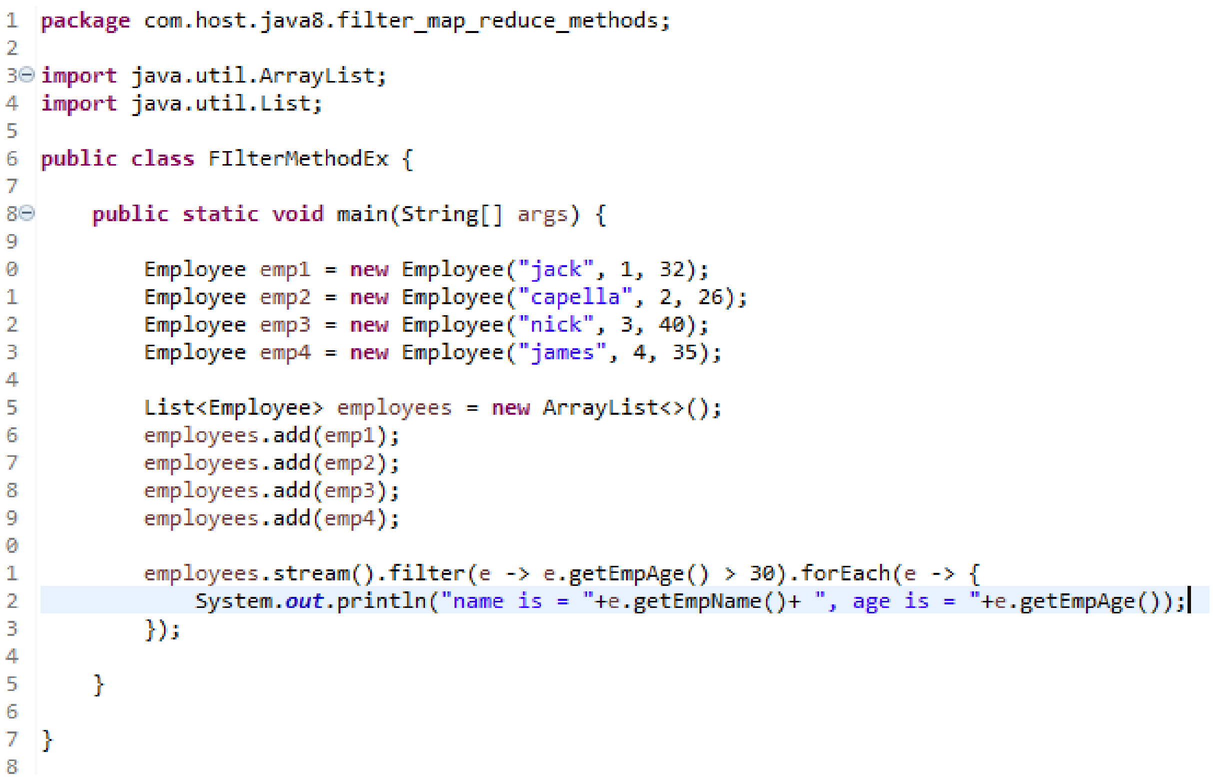
1. Filter
2. Map
3. flatMap
4. Sorted
5. Distinct
6. Peek
7. Limit
8. seek

Terminal Operations

1. forEach
2. Collect
3. Match
4. Count
5. Reduce
6. toArray
7. Min
8. Max
9. anyMatch
10. allMatch
11. findFirst
12. findAny
13. noneMatch

Can you show the employee details from the list, having age > 30 using java 8

By using filter() method we can achieve this

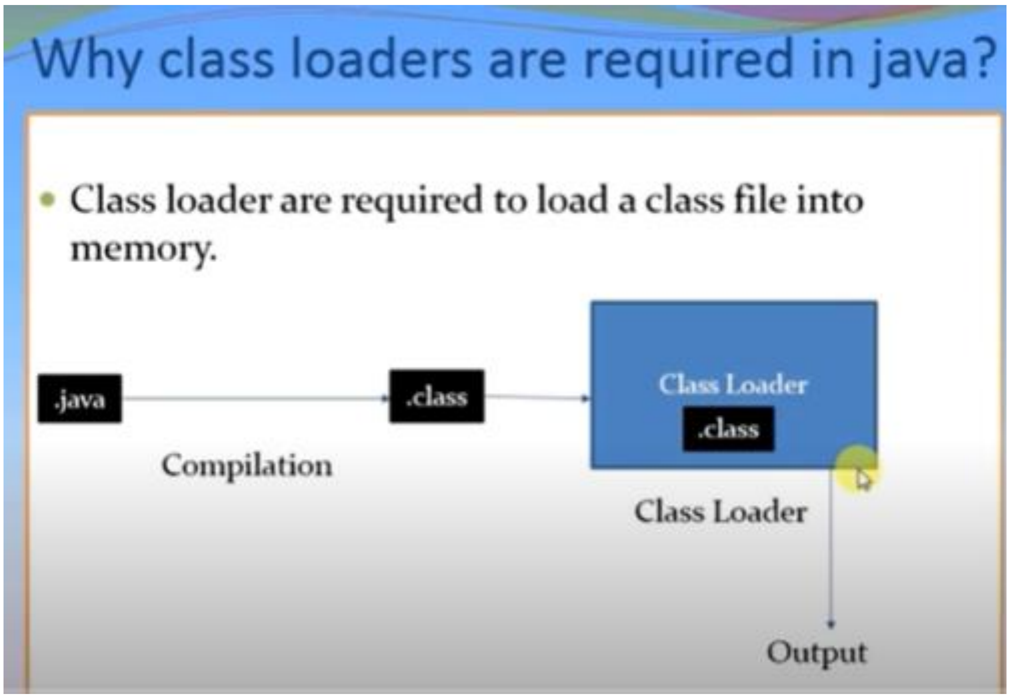


```java
package com.host.java8.filter_map_reduce_methods;

import java.util.ArrayList;
import java.util.List;

public class FIlterMethodEx {

    public static void main(String[] args) {

        Employee emp1 = new Employee("jack", 1, 32);
        Employee emp2 = new Employee("capella", 2, 26);
        Employee emp3 = new Employee("nick", 3, 40);
        Employee emp4 = new Employee("james", 4, 35);

        List<Employee> employees = new ArrayList<>();
        employees.add(emp1);
        employees.add(emp2);
        employees.add(emp3);
        employees.add(emp4);

        employees.stream().filter(e -> e.getEmpAge() > 30).forEach(e -> {
            System.out.println("name is = "+e.getEmpName()+ ", age is = "+e.getEmpAge());
        });

    }

}
```

Java Core Features

Class Loader

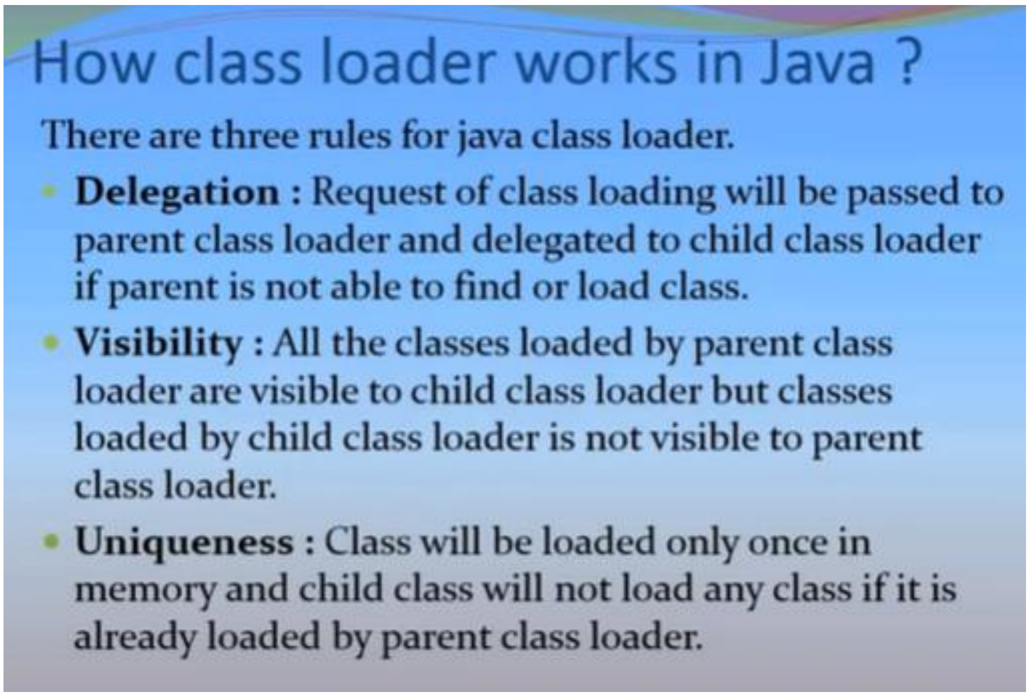
What is Loading:



Why class loaders are required in java?

- Class loader are required to load a class file into memory.

Characteristics

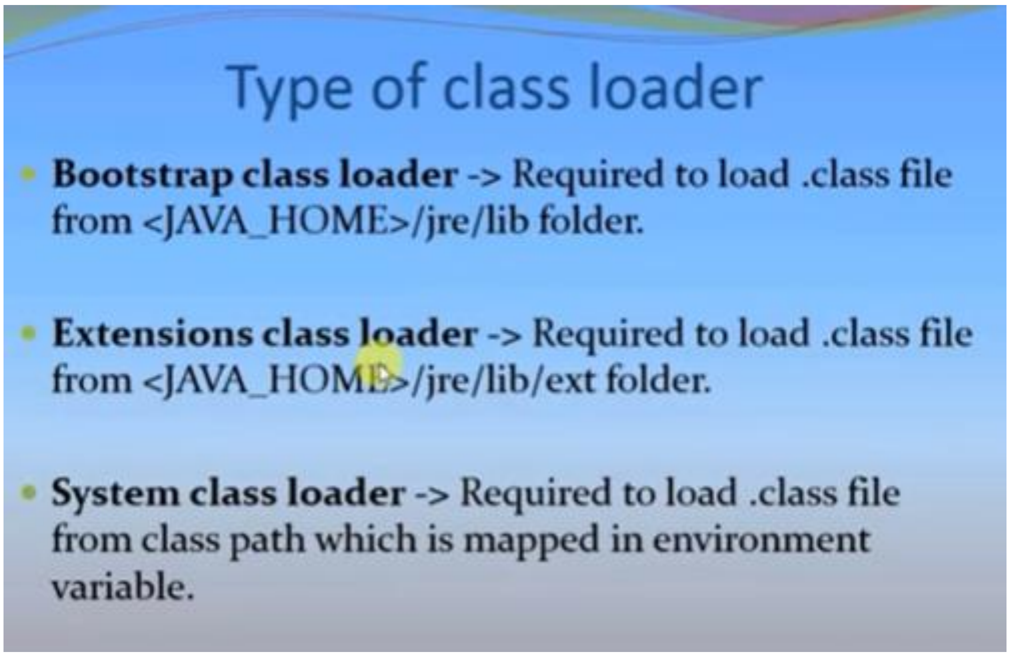


How class loader works in Java ?

There are three rules for java class loader.

- **Delegation :** Request of class loading will be passed to parent class loader and delegated to child class loader if parent is not able to find or load class.
- **Visibility :** All the classes loaded by parent class loader are visible to child class loader but classes loaded by child class loader is not visible to parent class loader.
- **Uniqueness :** Class will be loaded only once in memory and child class will not load any class if it is already loaded by parent class loader.

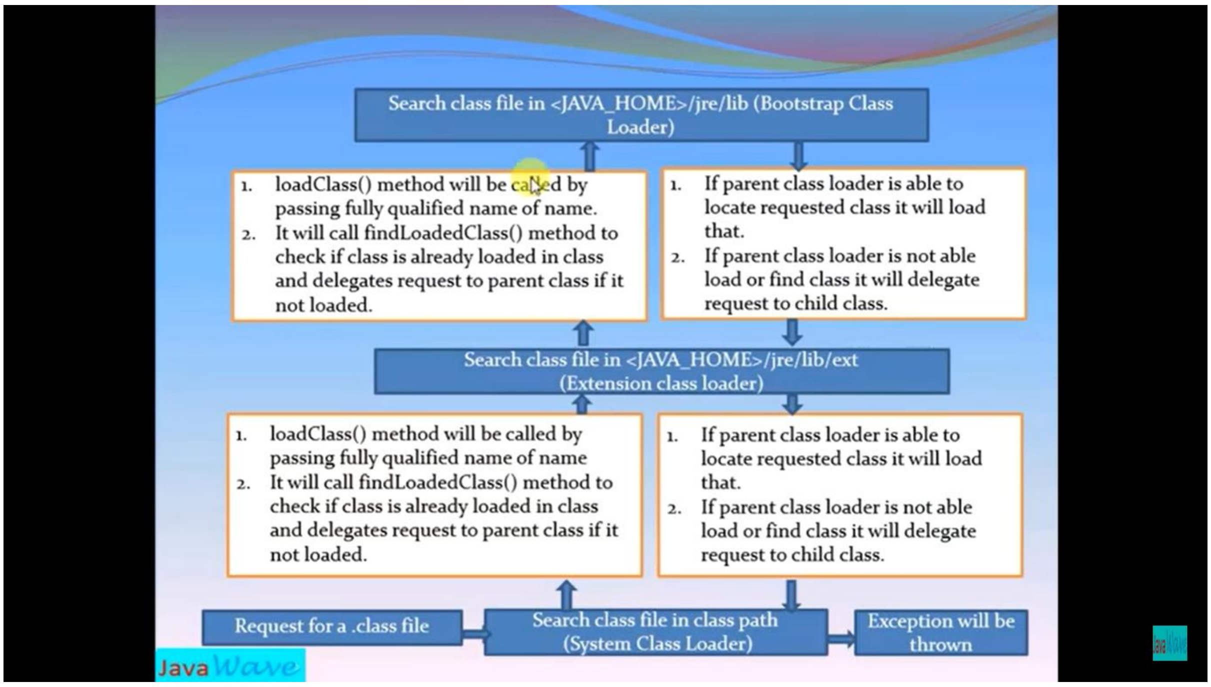
Types of Loaders:



Type of class loader

- **Bootstrap class loader** -> Required to load .class file from <JAVA_HOME>/jre/lib folder.
- **Extensions class loader** -> Required to load .class file from <JAVA_HOME>/jre/lib/ext folder.
- **System class loader** -> Required to load .class file from class path which is mapped in environment variable.

Process:



Search class file in <JAVA_HOME>/jre/lib (Bootstrap Class Loader)

1. loadClass() method will be called by passing fully qualified name of name.
2. It will call findLoadedClass() method to check if class is already loaded in class and delegates request to parent class if it not loaded.

1. If parent class loader is able to locate requested class it will load that.
2. If parent class loader is not able load or find class it will delegate request to child class.

Search class file in <JAVA_HOME>/jre/lib/ext (Extension class loader)

1. loadClass() method will be called by passing fully qualified name of name
2. It will call findLoadedClass() method to check if class is already loaded in class and delegates request to parent class if it not loaded.

1. If parent class loader is able to locate requested class it will load that.
2. If parent class loader is not able load or find class it will delegate request to child class.

Request for a .class file

Search class file in class path (System Class Loader)

Exception will be thrown

Whenever request comes to load .class file, first request will serve to System class loader

Step-1: search in memory from bottom to top approach

System Class Loader (it will search in memory)

If – not

Will delegate request to Extension-Class Loader (check in memory, if-not)

Will delegate to Bootstrap-Class-Loader

Step-2: Load .class file from top – lower approach

Boot strap loader (it will check in lib folder, if not available -> request delegates to extension class loader)

Extension Class Loader- (it will check in lib/ext folder, if -not available)

System class loader (it will check .class file based on class path) if not

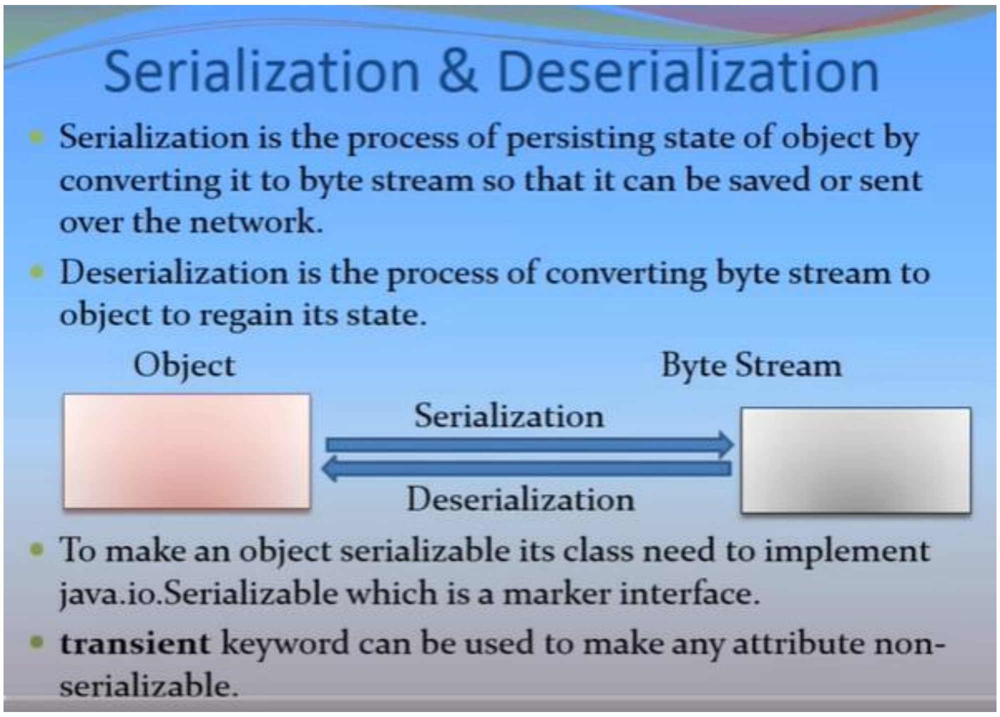
Throws NoClassFound Exception

Serialization & De-Serialization

Def:

Serialization: Converting object -> binary stream

De-Serialization: Converting Binary Steam -> Object form



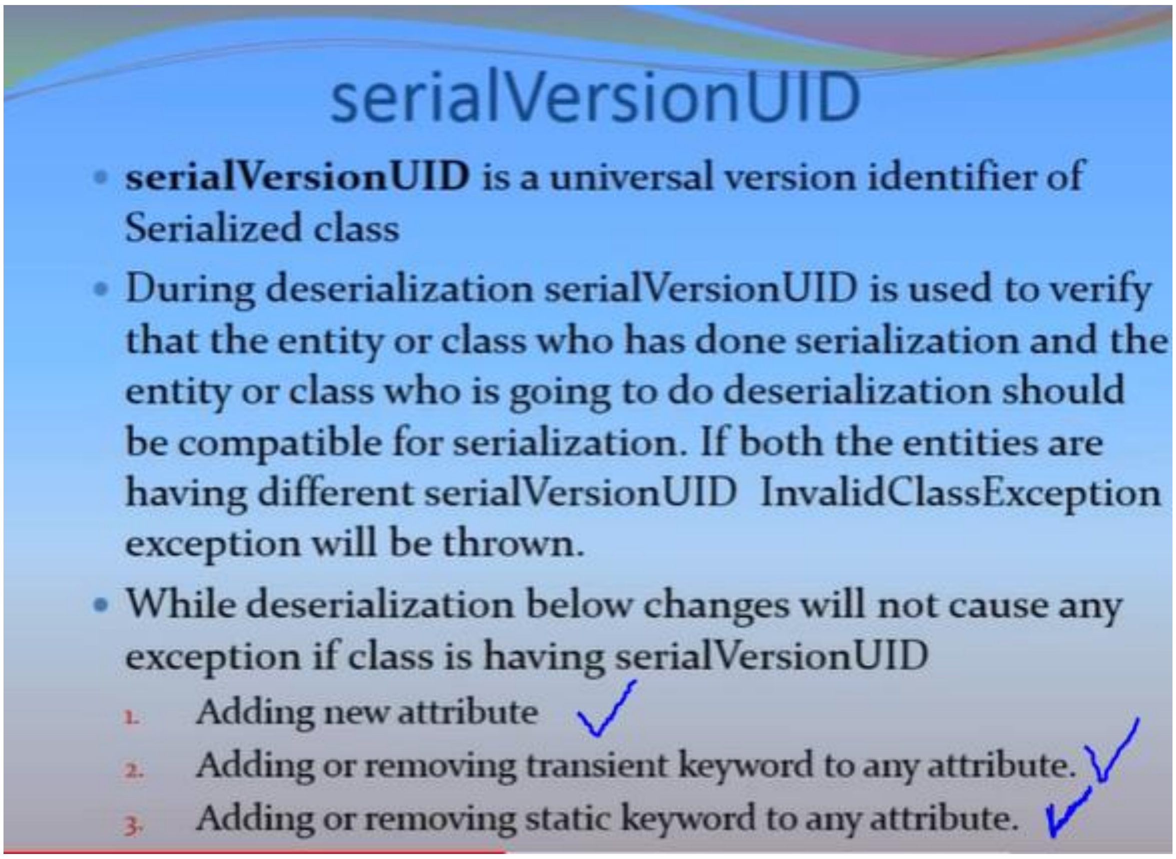
Serialization & Deserialization

- Serialization is the process of persisting state of object by converting it to byte stream so that it can be saved or sent over the network.
- Deserialization is the process of converting byte stream to object to regain its state.

Object
Byte Stream
Serialization
Deserialization

- To make an object serializable its class need to implement java.io.Serializable which is a marker interface.
- **transient** keyword can be used to make any attribute non-serializable.

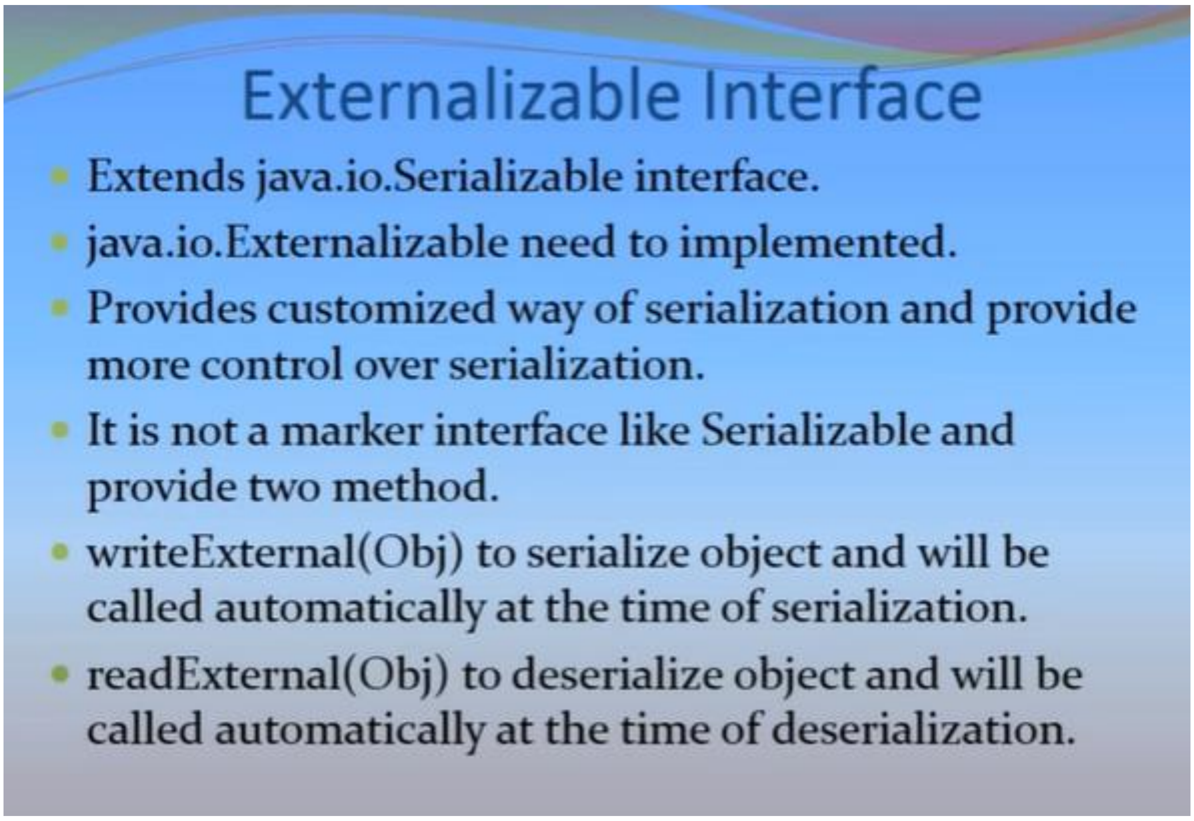
SerialVersionID



serialVersionUID

- **serialVersionUID** is a universal version identifier of Serialized class
- During deserialization serialVersionUID is used to verify that the entity or class who has done serialization and the entity or class who is going to do deserialization should be compatible for serialization. If both the entities are having different serialVersionUID InvalidClassException exception will be thrown.
- While deserialization below changes will not cause any exception if class is having serialVersionUID
  1. Adding new attribute ✓
  2. Adding or removing transient keyword to any attribute. ✓
  3. Adding or removing static keyword to any attribute. ✓

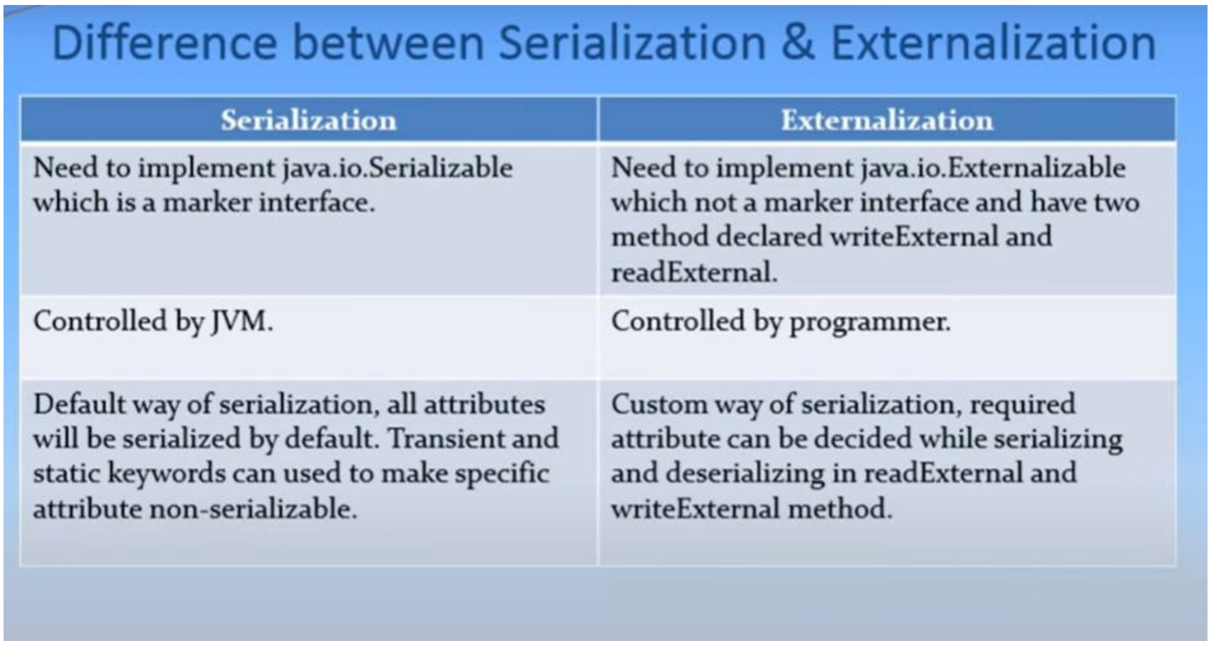
Externalizable Interface



Externalizable Interface

- Extends java.io.Serializable interface.
- java.io.Externalizable need to implemented.
- Provides customized way of serialization and provide more control over serialization.
- It is not a marker interface like Serializable and provide two method.
- writeExternal(Obj) to serialize object and will be called automatically at the time of serialization.
- readExternal(Obj) to deserialize object and will be called automatically at the time of deserialization.

Difference between Serializable and Externalizable interfaces

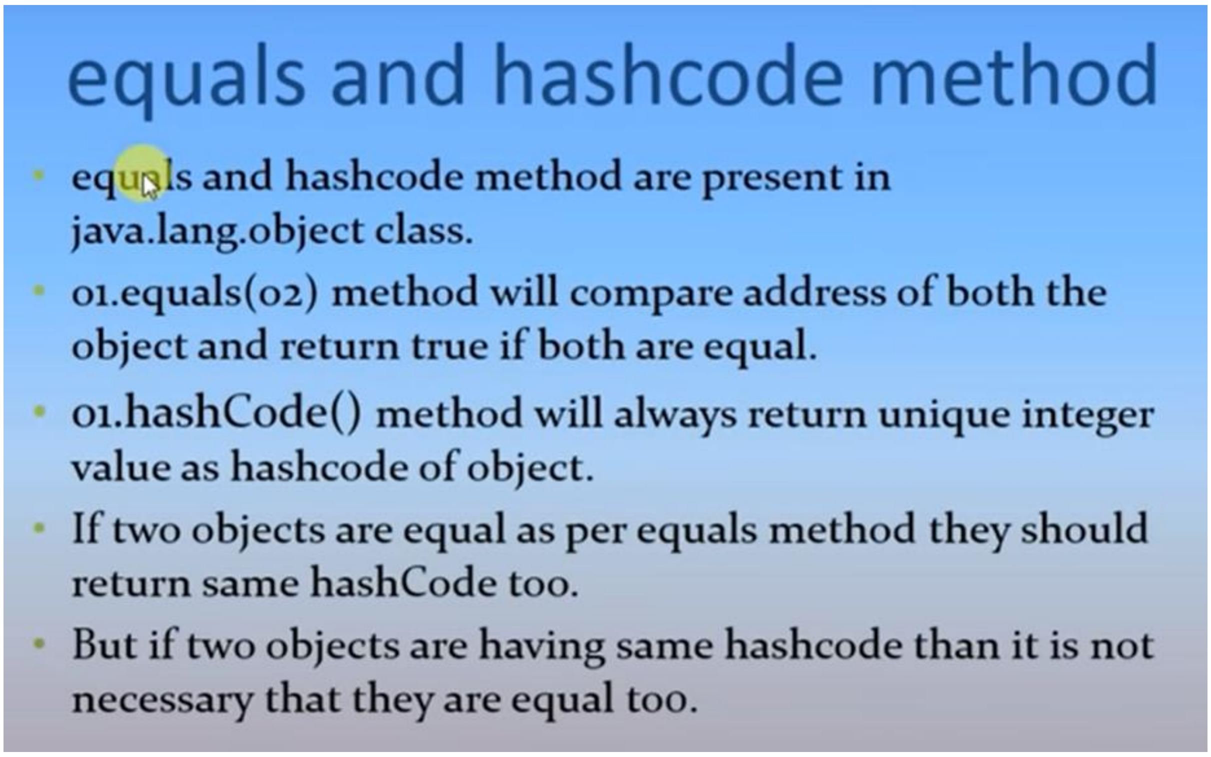


Difference between Serialization & Externalization

Serialization	Externalization
Need to implement java.io.Serializable which is a marker interface.	Need to implement java.io.Externalizable which not a marker interface and have two method declared writeExternal and readExternal.
Controlled by JVM.	Controlled by programmer.
Default way of serialization, all attributes will be serialized by default. Transient and static keywords can used to make specific attribute non-serializable.	Custom way of serialization, required attribute can be decided while serializing and deserializing in readExternal and writeExternal method.

HashCode () & Equals() methods

Why we use



equals and hashcode method

- equals and hashcode method are present in java.lang.object class.
- o1.equals(o2) method will compare address of both the object and return true if both are equal.
- o1.hashCode() method will always return unique integer value as hashcode of object.
- If two objects are equal as per equals method they should return same hashCode too.
- But if two objects are having same hashcode than it is not necessary that they are equal too.

Java Synchronization:

https://www.javatpoint.com/synchronization-in-java#:~:text=Synchronization%20in%20java%20is%20the,to%20access%20the%20shared%20resource.

Synchronization in java is the capability to control the access of multiple threads to any shared resource

Java synchronization is the better option where we can allow only one thread to access a shared resource.

Why we use?

To prevent Thread Interference

Types of Synchronization

1. Process Synchronization
2. Thread Synchronization

For thread synchronization refer example in workspace

Destructor

What it is?

Destructor is special method, it will execute automatically once object life cycle is finished

A Destructor is also called as “”de-allocate and free memory”

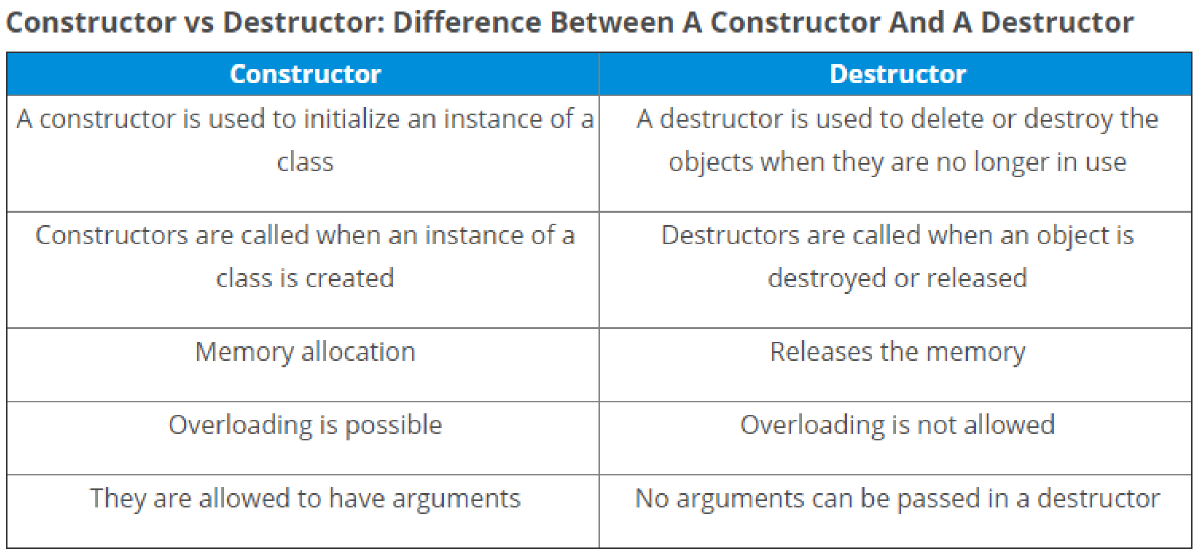
What it does?

1. Releasing the release locks [Ex: Synchronization and locking the shared resources while it is accessing by multiple threads]
2. Closing all database connections and files
3. Releasing the network resources
4. Recovering heap space allocated during the lifetime of an object

When it will execute?

It is executed when object is destroyed and it is the last method executed by class

Constructor vs Destructor



Constructor vs Destructor: Difference Between A Constructor And A Destructor

Constructor	Destructor
A constructor is used to initialize an instance of a class	A destructor is used to delete or destroy the objects when they are no longer in use
Constructors are called when an instance of a class is created	Destructors are called when an object is destroyed or released
Memory allocation	Releases the memory
Overloading is possible	Overloading is not allowed
They are allowed to have arguments	No arguments can be passed in a destructor

Garbage Collector

What it is?

Garbage collector is framework/software which runs on the Java Virtual Machine to recover the memory by deleting objects which are no longer in use or have finished their life cycle

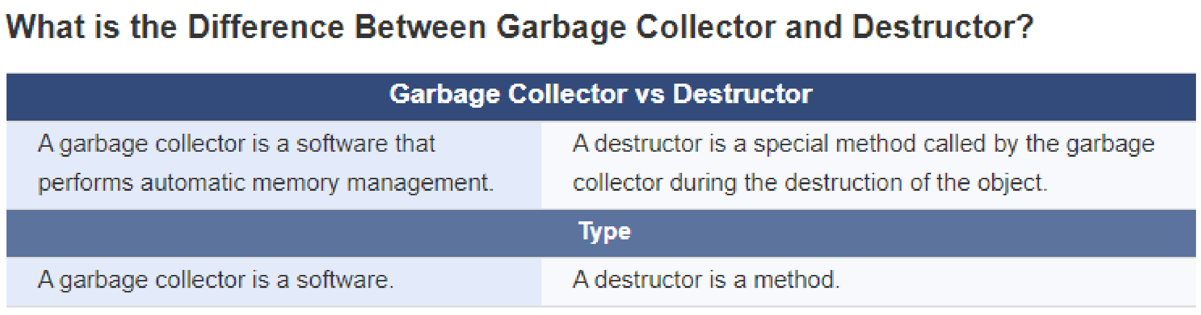
How it works?

1. Garbage collection is mainly the process of identifying unreachable objects and deleting them from memory
2. The implementation lives in the JVM
3. Types of garbage collectors in java
    a. Serial Garbage Collector
    b. Parallel/Throughput Garbage Collector
    c. CMS Collector
    d. G1 Collector

Advantages

1. It automatically deletes the unused objects that are unreachable
2. Garbage collection makes java memory efficient
3. It need not be explicitly called since the implementation lives in JVM
4. Garbage collection has become more important and standard component in other programming languages also

Garbage Collector vs Destructor



What is the Difference Between Garbage Collector and Destructor?

Garbage Collector vs Destructor	
A garbage collector is a software that performs automatic memory management.	A destructor is a special method called by the garbage collector during the destruction of the object.
Type	
A garbage collector is a software.	A destructor is a method.

Object Oriented Language

Objects contains the data inti form of attributes and code in the form of methods

Procedural Language:

1. Programming derived from Structured programming concepts like procedures
2. Procedures also known as routines, sub routines or functions
    a. Simply consists of series of computational steps
3. During programming execution any given procedure might be called at any point

Where you have used OOPs concepts in Automation

Abstraction:

Abstraction is methodology of hiding the implementation of internal details and showing the functionality to the users

Ex:

In Page object model design pattern, we write locators (such as ID, Name, Xpath,) in a page class

We utilize those locators in tests, but we can't see these locators in tests. Literally we hide the locators from the tests.

We can achieve Abstraction through interfaces and abstract classes

Interface:

As we all know assigning Driver() class object to WebDriver interface

We can achieve 100% abstraction and multiple inheritance in Java with Interface.

Inheritance:

We create Base Class in framework to initialize WebDriver instance, WebDriver waits, Property Files

We extend this base class in other test class to utilize these features.

Polymorphism:

We can decide the test type at run time based on the parameter we are sending from the test suite

So based on the test type we can choose the web, api or Excel client driver initialization so called polymorphism

Method Overloading:

A class having multiple methods with same with different arguments called method overloading

Implicit wait in selenium is example of overloading, In implicit wait we use different time stamps such as

1. SECONDS
2. MINUTES
3. Hours

Method Overriding:

Implementation different business logic in child classes for the methods created in parent class

Ex: Navgate() , refresh(), maximize(), methods .. selenium driver classes

Encapsulation:

QA Managerial

By what factors you can measure quality of test execution?

Defect Rejection Ratio: (Total No of Defects Rejected / Total No of Defects raised) X 100

Bug Leakage Ratio: (Total No of Defects Missed / Total No of Software Bugs) X 100

How do you resolve team conflicts?

We definingly expect and prepare for conflicts during the test project

1. Keep communication open to share the conflicts and gaps in the team
2. Will explain the importance of team work and co-operation for the success of the project
3.