Contents

- Singleton Design Pattern
- Abstract Class , Interface (Example of Interface : Runnable Interface) , Anonymous Inner Class
- Functional Interfaces , Lambdas
- Generics , Streams

SINGLETON DESIGN PATTERN :

Singleton pattern is a design pattern which restricts a class to instantiate its multiple objects. It is nothing but a way of defining a class. Class is defined in such a way that only one instance of the class is created in the complete execution of a program or project. It is used where only a single instance of a class is required to control the action throughout the execution.

Challenge 1 : We have a Person class . Our task is to make sure that when we create objects of Person class , constructor gets called only once, i.e. all objects have the same hashcode .

Challenge 2 : Why do we need Singleton classes ? (HINT : Driver Class for making connections to the database .)

Challenge 3 : Why does the newly created getPerson function in Person needs to be static ?

References :

https://www.geeksforgeeks.org/singleton-design-pattern-introduction/

https://www.geeksforgeeks.org/singleton-design-pattern/

https://www.geeksforgeeks.org/java-singleton-design-pattern-practices-examples/

ABSTRACT CLASS :

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

```
abstract class Shape
{
    int color;

    // An abstract function
    abstract void draw();
}
```

Challenge 1 : Can we have a function in the abstract class with a body ?

References :

https://www.geeksforgeeks.org/abstract-classes-in-java/

INTERFACE :

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship . That means all the methods in an interface are declared with an empty body and are public and abstract and all fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface.

```
// A simple interface

interface Player

{

    final int id = 10;

    int move();

}
```

Challenge 1 : Can we have a function in the interface with a body ? (HINT : default function)

Challenge 2 : Can we override the default function in the implementing class ? OR How do we call the default methods of an interface in the implementation class ?

HINT : If two interfaces have same name default method , we need to override that method in our implementation class . Its a rule .

We override the default function in the class and inside its body use the <interfaceName>.super.<methodName>()

Challenge 3 : If we are overriding a default function in our implementation class , then how can we call the default function of the interface ?

HINT : <InterfaceName>.super.<functionName>(x,y)

Challenge 4 : How to access the variables in our implementation class which have been defined in the interface ? Can we change its value in the implementation class and Why ? (HINT : All variables in interface are public, final and static)

Challenge 5 : In the implementation class , in the implementation of an abstract method ,can we give access modifier as protected ?

Challenge 6 : If a class A is implementing an interface I and extending a class C . Which is correct ?

class A extends C implements I

OR

class A implements I extends C

Challenge 7 : If my class A extends interface B and interface C , both having same default method “power” . Even if my object of class A does not call power and does not overrides the default method power , will there be any issues and why ?

Challenge 8 : Is it possible that in parent class , function A has protected access modifier In child class which extends parent , while overriding , can we give access modifier of A as :

(i) public

(ii) private

HINT : we can't assign weaker privileges while overriding methods .

References :

https://www.geeksforgeeks.org/interfaces-in-java/

https://www.geeksforgeeks.org/static-method-in-interface-in-java/

https://www.geeksforgeeks.org/default-methods-java/

ANONYMOUS INNER CLASS :

An anonymous inner class can be useful when implementing an interface or abstract class with certain “extras” such as overriding abstract methods of interface/abstract class , without having to actually subclass a class.

Challenge 1 : Is multiple inheritance allowed in Java with (i) classes (ii) interface ?

Challenge 2 : Is multiple inheritance allowed in Java ?

Challenge 3 : Does an interface implements other interface or extends other interface ?

Challenge 4 : Can an interface extend a class and why?

Challenge 5 : What is the difference between Abstract Class and Interface ?

Challenge 6 : Can we override default methods of an interface in other interface which extends it ?

HINT : https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/

References :

https://www.geeksforgeeks.org/nested-classes-java/

https://www.geeksforgeeks.org/anonymous-inner-class-java/

https://stackoverflow.com/questions/2515477/why-is-there-no-multiple-inheritance-in-java-but-implementing-multiple-interfac

https://www.geeksforgeeks.org/java-and-multiple-inheritance/

FUNCTIONAL INTERFACE :

A functional interface is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. Runnable, ActionListener, Comparable are some of the examples of functional interfaces.

Challenge 1 : Lets checkout Runnable Interface

Challenge 2 : Let us now create an object of Runnable interface using Anonymous Inner Class

Challenge 3 : Lets now try to write our first lambda by trying to do the Challenge 2 using Lambda .

Challenge 4 : Now we have our object of Function Interface (Runnable) . How do we know which function to call ?

Challenge 5 : Lets now create a custom Function Interface with a function taking 2 argos . And now lets create object of this custom functional interface using (i) Anonymous Inner Class (ii)Lambda

Challenge 6 : Convert the following to Lambda :

```
FuncInt funcInt1 = new FuncInt() {
    @Override
    public int add(int a, int b) {
        System.out.println("Got the arguments as " + a + " " + b);
        return a + b;
    }
};
```

Challenge 7 : If the interface for which we want to write lambda contains 2 abstract methods . Is there a problem ?

Challenge 8 : If the interface for which we want to write lambda contains 2 default methods and a single abstract . Is there a problem ? How can we call the default methods ?

Challenge 9 : Why do we use @FunctionalInterface annotation ? Is it mandatory ?

HINT : @FunctionalInterface tells users that this interface is intended to be a functional interface and if there is more than one abstract methods added , then it throws at the interface level at compile time .

Challenge 10 : Is Comparator Interface a Functional Interface ?

HINT : equals method is coming from Object class .

Challenge 11 : If we are creating lambda for a Functional Interface in a class . Does that class need to implement the interface?

References :

https://www.geeksforgeeks.org/functional-interfaces-java/

https://stackoverflow.com/questions/50892117/why-to-use-functionalinterface-annotation-in-java-8

https://piyush5807.medium.com/functional-interfaces-in-a-nutshell-for-java-developers-54268e25324

GENERICS :

Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Challenge 1 : Can we convert our custom Functional Interface to be a generic interface with input type as T and return type as R .

HINT : We need to also convert our abstract method accordingly .

```
@FunctionalInterface
public interface FuncInt<T, R> {

    R add (T a, T b);

    boolean equals(Object obj);

    default int subtract(int a, int b) { return a - b; }

    default int multiply(int a, int b) { return a * b; }
}
```

Now when we create the lambda for this functional interface , we will also need to accommodate the generic types we want . *Below are 2 examples of how we can use the same Interface with different lambdas because our input type and output type data types are generics .*

```
FuncInt<Integer, String> funcInt = (c, d) -> {
    System.out.println("Got the arguments as " + c + " " + d);
    return String.valueOf(c + d);
};
```

```
FuncInt<String, String> funcInt = (c, d) -> {
    System.out.println("Got the arguments as " + c + " " + d);
    return c + d;
};
```

References :

https://www.geeksforgeeks.org/generics-in-java

STREAMS :

A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. stream() is sequential processing one by one and parallelStream is multithreading .

Intermediate Operations:

1. map: The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.
2. filter: The filter method is used to select elements as per the Predicate passed as argument.
3. sorted: The sorted method is used to sort the stream.

Terminal Operations:

1. collect: The collect method is used to return the result of the intermediate operations performed on the stream.
2. forEach: The forEach method is used to iterate through every element of the stream.
3. reduce: The reduce method is used to reduce the elements of a stream to a single value. The reduce method takes a BinaryOperator as a parameter. Reduce does not return a Stream .

stream() *parallelStream()*

Challenge 1 : Given a list of city names .We have to find out which cities start with a vowvel and return a list with cities which start with vowels and are in uppercase .

Challenge 2 : You have a list of integers, you need to find the sum of squares of even numbers [1, 2, 3, 4, 5, 6, 7, 8] = [4 + 16 + 36 + 64 = 120]

Challenge 3 : You have a list of cities , you need to concatenate them .

Challenge 4 : You have a list of cities, you need to concatenate them but with a space in between .

Challenge 5 : In the last scenario , what will be the result if we use identity as null ?

Challenge 6 : You have a list of cities, you need to sort them in :

(i)ascending order (ii)descending order (iii) Increasing order of String length (iv) decreasing order of String length

Challenge 7 : What is the difference between IntStream and Stream<Integer>?

HINT : https://stackoverflow.com/questions/64974871/what-is-the-difference-between-intstream-and-streaminteger

ASSIGNMENT : Assignment 2 (Streams)

MUST Watch Resources for Lambda and Streams :

https://www.youtube.com/watch?v=F73kB4XZQ4I

https://www.youtube.com/watch?v=1OpAgZvYXLQ

References :

https://piyush5807.medium.com/declarative-programming-in-java-using-streams-and-lambdas-3f71edcd7a74 IMP

https://stackify.com/streams-guide-java-8/

https://www.geeksforgeeks.org/stream-in-java/

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

https://www.geeksforgeeks.org/parallel-vs-sequential-stream-in-java/

https://www.geeksforgeeks.org/streams-arrays-java-8/