SOEN 387
WEB-BASED ENTERPRISE APPLICATIONS DESIGN

TUTORIAL – 1

Java 8 Features

Agenda

- Java 8 features
- Constructor Chaining
- Nested classes in Java
- Inner Class -> Regular vs Static Inner class
- Anonymous Class
- Wildcards In Generics
- Lambda Expression

Features of Java8

LANGUAGE-LEVEL SUPPORT FOR LAMBDA EXPRESSIONS

INTERFACE DEFAULT AND STATIC METHODS

Constructor Chaining

- Calling constructor from another constructor
- Using **this()** keyword for constructors in same class
- using **super()** keyword to call constructor from base class

Q) To create your own Exception class with a getMessage method in it

```
public class MyException extends Exception{
    private int code;
    MyException(int code){
        super();
        super.getMessage();
        this.code=code;
        System.out.println(getMessage(code));
    }

    private static String getMessage(int code){
        switch (code){
            case -1: return "invalid number";
            case -2: return "invalid userId";
            default: return "success";
        }
    }

    public static void main(String[] args) {
        try{
            throw new MyException(-1);
        }
        catch (Exception e){
            System.out.println("Exception thrown: " + e.getMessage());
        }
    }
}
```

OUTPUT

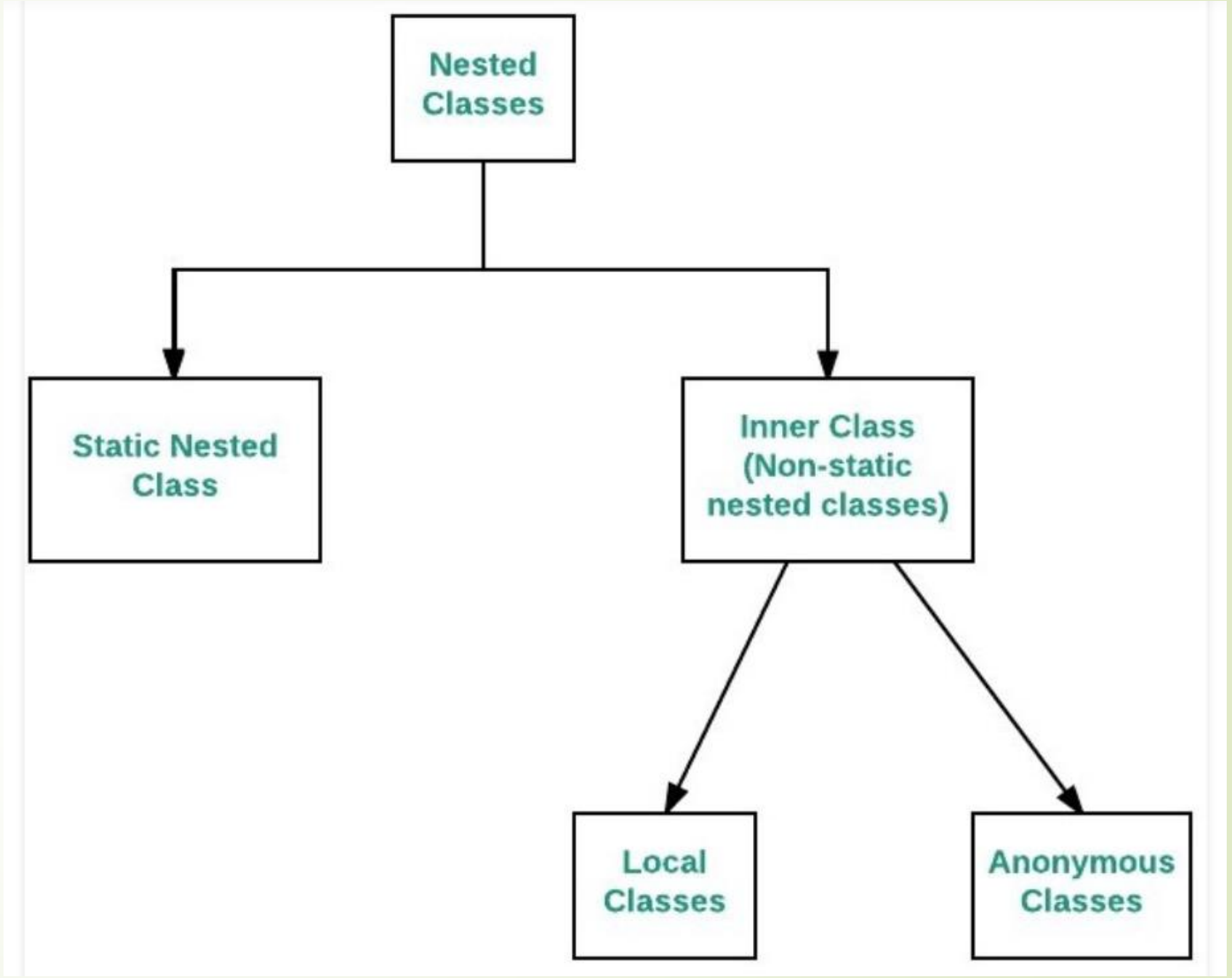
```
invalid number
Exception thrown: null
```

Points to remember

For MyException class

- Extend Exception class
- Constructor call using super keyword
- Function should be static

Nested Classes in Java



Inner Class

- Inner class suggests a class within an another class

```
package week1;

class OuterClass{
    private int a=10;
    class NestedInnerClass{
        int b=20;
         void show(){
            System.out.println("The value of outer int is "+a+" and the value of inner int is "+b);
        }
    }
}
public class InnerClassExample {
    public static void main(String[] args) {
        OuterClass.NestedInnerClass obj=new OuterClass().new NestedInnerClass();
        obj.show();
    }
}
```

OUTPUT

```
The value of outer int is 10 and the value of inner int is 20
```

Static Inner class

- It is considered as a static member of outer class

```
class Outer{
    static class Inner{
        public static void printMessage(){
            System.out.println("Hello from Inner Static class");
        }
    }
}
public class StaticInnerExample {
    public static void main(String[] args) {
        Outer.Inner obj=new Outer.Inner();
        obj.printMessage();
    }
}
```

OUTPUT

```
HEllo from Inner Static class
```

Anonymous class

- Its an inner class without a name

```
interface MyInterface{
    void showMessage();
}
public class AnonymousClass {
    public static void main(String[] args) {
        MyInterface obj=new MyInterface() {
            @Override
            public void showMessage() {
                System.out.println("We are learning Anonymous Inner Class");
            }
        };
        obj.showMessage();
    }
}
```

OUTPUT

```
We are learning Anonymous Inner Class
```

Wildcards in Generics

SYMBOL	MEANING	SIGNIFICANCE
?	Unbounded wildcard	The family of all types.
? extends **SuperType**	Wildcard with an upper bound	The family of all types that are SuperType itself or subtypes of SuperType.
? super **SubType**	Wildcard with a lower bound	The family of all types that are SubType itself or supertypes of SubType

Lambda Expressions

- Function Interface
- Zero Parameter

```
interface MyLambda{
    void show();
     default void printMsg(int x){
        System.out.println(2*x);
    }
}
public class LambdaClass {
    public static void main(String[] args) {
        MyLambda obj=()->System.out.println("Hello! It's a tutorial class");
        obj.show();
        obj.printMsg( x: 10);
    }
}
```

OUTPUT

```
Hello! It's a tutorial class
20
```

Lambda Expression with parameters

```
interface MyLambda2{
    String printMessage(String message);
}
interface  MyLambda3{
    int add(int x,int y);
}

public class LambdaClass2 {
    public static void main(String[] args) {
        MyLambda2 obj=(Msg) ->  "Hello! "+ Msg ;
        System.out.println(obj.printMessage("Welcome to Tutorial 1 "));

        MyLambda3 obj2=(a,b)->a+b;
        System.out.println(obj2.add( x: 10, y: 20));
    }

}
```

OUTPUT

```
Hello! Welcome to Tutorial 1
30
```

Exercise

- Given a Student Class with first name, last name and id
- Apply the following sorting on array of students:

1. Sort by Student id
2. Sort by FirstName
3. Sort by lastname then by firstname

```
public class Student {
    public long id;
    public String firstName;
    public String lastName;
}
```

Hint: Function Reference

- Old Java:

  public static void java.util.Arrays.sort(Object[] a, Comparator c)

- New Java:

  public static <T> void java.util.Arrays.sort(T[] a, Comparator<? super T> c)

Refer:
https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html
https://www.tutorialspoint.com/java/util/arrays_sort_super.htm
https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Solution

```
public static void main(String[] args) {
        Student[] students = new Student[1000];
        // load students from file or other data source

        // Sort based on Id
        Arrays.sort(students, (a, b) -> a - b);

        // Sort based on FirstName
        Arrays.sort(students, (a, b) -> a.firstName.compareTo(b.firstName)); // problem nulls!
        Arrays.sort(students, (a, b) -> ("" + a.firstName).compareTo("" + b.firstName)); // handling nulls

        // Alternatively handling nulls differently
        Arrays.sort(students, (a, b) -> (a, b) -> a.firstName == b.firstName? 0:
                a.firstName == null? -1: // nulls precede empty strings
                        b.firstName == null? 1:
                                a.firstName.compareTo(b.firstName));

        // Sort based on LastName and then firstName
        Arrays.sort(students, (a, b) -> {
            int ck = ("" + a.lastName).compareTo("" + b.lastName); // lastName
            return ck != 0? ck:
                    ("" + a.firstName).compareTo("" + b.firstName); // then firstFame
        });
}
```

References

- http://www.differencebetween.net/technology/difference-between-java-7-and-java-8/#targetText=Java%208%2C%20on%20the%20other,enhancements%20to%20the%20Java%20model.&targetText=Java%208%20is%20a%20major,programming%20called%20the%20Lambda%20Expressions.
- https://www.geeksforgeeks.org/anonymous-inner-class-java/
- https://www.geeksforgeeks.org/nested-classes-java/
- Difference between Java 7 and Java 8 | Difference Between http://www.differencebetween.net/technology/difference-between-java-7-and-java-8/#ixzz5yguBEe8q
- https://www.geeksforgeeks.org/constructor-chaining-java-examples/