Java

- **Private class**

  Inner classes can be private.

  We cannot declare top level class as private

In eclipse,

- If the filename is classname.java, then that class should have main method.

Nested Classes in Java (Article - GfG)

Nested classes in java

nested class - inner class

enclosing class - outer class

① nested ~~inner~~ class has access to members (including private) of outer class.

② Outer class does not have access to the members of the nested ~~inner~~ class.

③ nested class is a member of outer class.

So, nested class can have any access modifier

Nested classes — static nested
— inner (non-static) — Local
— Anonymous

normal inner class (non-static)

- to create its object, outer class object has to exist.
- both static & non-static members of outer class can be accessed directly.

Static nested class object

- can exist without outer class object
- can access static members of outer class.

Static nested class

```
Outerclass.StaticNestedClass nestedObject =
        new Outerclass.StaticNestedClass();
```

Accessing an inner class (non-static)

* `OuterClass outerObject = new OuterClass();`

`OuterClass.InnerClass innerObject = outerObject.new InnerClass();`

`outerObject.new InnerClass();`

* Inside normal/regular inner class, static members can't be declared.

* Method Local Inner Class

- Inner class declared within a method of an outer class

Inner — nested inner
- method local inner
- static nested
- anonymous inner.

Anonymous Class

```
class Demo {
    void show()
    {
        ---
    }
}
```

```
class Haver1Demo {
    static Demo d = new Demo() {
        void show()
        {
            super.show();
            ---
        }
    };
    main() {
        d.show();
    }
}
```

Anonymous class - can't have explicit constructor

type Syout + backspace → eclipse shortcut for System.out.println()

sc.next() ← if you press enter, "" value assigned

Doubt - ?

Check data type in Java

```
int x=100;
System.out.println(((Object) x).getClass().getSimpleName());
```

This method is callable by objects only.

So, we need to cast the primitive data type to Object first..

- ```
  String str = "";
  System.out.println(str.getClass().getSimpleName());
  ```

If you don't initialize, you'll get the error → error: Local variable may not have been initialized.

- Integer.parseInt( )
- split() - this method returns a string array
  ```

Exception handling

We can't have catch or finally clause without a try statement.

We can throw without try-catch.

→ this will be handled by default catch mechanism

HashMap (GfG) | HashMap in java

- basic implement^n of Map interface of java
- key value pairs
- ~~If you~~ duplicate key → element of corresponding key will be replaced.
- key can be null. (but only one null key object) any no. of null values are allowed.
- import java.util.HashMap;
- HashMap<String, Integer> map = new HashMap<>();
  - map.put("swarali", 10);
  - map.size();

  or

  HashMap<String, Integer> map = new HashMap<String, Integer>();
- System.out.println(map); //Printing elements in object of map
- Integer a = map.get("swarali");
- if(map.containsKey("swarali"))

→ {swarali=10}

Load factor - measure that decides when to increase the capacity of the Map
- The default load factor is 75% of the capacity

HashMap<Integer, String> hm1 = new HashMap<>(10);

Initial capacity

= new HashMap(3, 0.5f)

Initial capacity — Load factor

Insertion order is not retained in hashmap

Internally # hash is generated for every element.

key

hm.remove(4); remove element with a key

Iterate the map using for each loop

```
for(Map.Entry<String, Integer> e : map.entrySet())
    System.out.println(e.getKey() + " " + e.getValue());
```

Spring framework

Spring framework

PAGE No. / DATE / /

What Is Spring framework All about? (Youtube - JavaBrains)

~~All About~~

Frameworks - provide structure & common pattern to make easy building applications

* network of objects initializ^n.
  objects are related

sharing multiple existing instances - needed in business services

* Types of objects - Objects that hold data,
  instances with business logic methods
  only 1 instance is needed (with functionality)

* Spring Application context - manages object instances
* Dependency injection - every object has instances to all other instances it requires

Applications need database connectivity

JDBC - to work with databases in java
unpleasant

* Spring → comes with data access APIs & mechanism for connectivity, querying, and dbms and more
  - better experience on

Spring → Application context and dependency injection

basic functionalities → data access

→ Spring MVC

PAGE NO.

DATE / /

✓ enterprises expose REST APIs to be consumed by other applic°

So, spring comes with (Spring MVC) web framework

create web apps & REST APIs

* lessons learned from Codechef questions

```
import java.util.Arrays;

int[] arr = new int[5];

Arrays.fill(arr, 0);
```

all elements as 0.

- Comparing strings with == gave me wrong answer. So, use equals().

→ s1.equals(s2)

Don't accept String immediately after accepting integer.

① Accept integer
② sc.nextLine();
③ Accept string

Lambda expressions and Functional Interfa

Lambda expressions (Youtube - Telusko)

↳ Java 8 launched in 2014.

class → object

declare behavior in interface and define behavior in class

Functional interface - only one abstract method

Marker interface - no method

@FunctionalInterface

↳ use to write this annotation before interface.

```
@FunctionalInterface
Interface A {

    void show();

    String toString(); ← this is allowed because every class extends Object class which has this method

}

A obj = new A {                          ← Anonymous class
    Public void show() {
        System.out.println("Hi");
    }
}

obj.show();
```

↑ Instead this, use Lambda expression

→ PTO

⑨ Functional Interface

```
interface A
{
    void show();
}
```

```
public class Demo
{
    public static void main(String[] args)
    {
        A obj = () ->
        {
            System.out.println("Hi");
        };

        obj.show();
    }
}
```

parameters

no need to write method name here because functional interface ~~will have~~ has only one method

Iterator interface

Iterator interface

- known as universal cursor
- to iterate over a collection of java object components
- appropriate for all the classes of collection framework
- If collection is modified while traversing over an iterator, iterator will throw an exception when user will attempt to get the next component from the iterator.
- boolean hasNext()

  next()

  default void remove()
- Trying to remove items using for or for-each loop because collection is changing size

```
ArrayList<String> cars = new ArrayList<String>();
cars.add(_ _)

Iterator<String> it = cars.iterator();
System.out.println(it.next());

Iterator it iterator = cityNames.iterator();
```