Static var, fun, ~~class~~ block ★★★.

Unit - 2

- static KW can be used with var, fun, ~~class~~, block, class & import

Static var → All obj will share same copy of the static var

Static fun → St. fun can be accessed w/o creating an obj. It can be called directly using class name.

Static block → St. block is similar to a const^{r}.

- It runs auto^{y} when a class is first used.
- It can be used to initialize the var.

Q) Why is st. block used, if it is similar to const^{r}?

A) When St. KW is used, then obj is not created. So const^{r} will not be called. ∴ St. block is used.

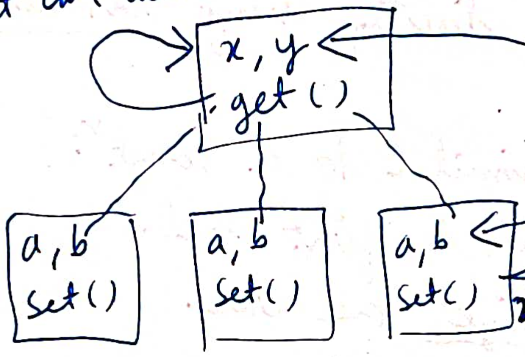
Rule w/ st. fun

- St. fun can call only other st. var & fun.
- Non st. fun can access all " var & fun.

eg var x, y r static
" a, b r normal.

fun set is normal - it can access all var a, b, x, y
" get " static - " " " only static var
x & y.

1. get can access only static var



2. set can access st. & non st. variables

//all obj have same copy of x, y (Static)
" " " diff " " a, b (Normal).

```
class box {
    protected static int l, w, h; //st. var.
    Static {
            l=0; w=0; h=0;    //static block.
    }

    public static void set(int x, int y, int z) {
            l=x; w=y; h=z;    //st. fun
    }
    public static void get() {
            Sopln(l); Sopln(w); Sopln(h);   //st. fun.
    }
}
C D { P S V m (..) {
    //No need to create obj of box.
    //Static fun can be called using Class name
      box.set(1, 2, 3);
      box.get();
  }
}
```

Nested Classes

- A class created / defined inside another class is called a nested class.
- Inner class can <u>directly access</u> var of outerclass but outerclass can't " " " " innerclass.
- Outerclass need to create obj of innerclass to access its variables.

PV → public void

```
outerclass {

        PV  outerfun() {
                    // define
        }

        class innerclass {
                    PV  innerfun() {
                            // define
                    }       outerfun();   → innerclass can
        }                                   directly call
                                            outerfun.

        innerclass  obj = new  innerclass();
          obj. innerfun();  → outerclass needs to
                              create obj of innerclass
                              to call innerfun.
}
```

can't call innerfun directly.

Creating obj of innerclass in demo class

- obj of a simple class is created inside main()

  eg box obj = new box();

- However, obj of an innerclass can't be created directly
- I st we need to create obj of outer class. Then we can create obj of innerclass using outer class.

eg PV → public void

```
class outerclass {
        PV outerfun() {
                Sopln("outerfun");
        }

    class innerclass {
            PV innerfun() {
                    Sopln("innerfun");
            }
    }
}
```

```
c D { p s v m (...) {
// to create obj of innerclass, Ist we need to create obj
   of outer class.
create outer obj ← outerclass obj1 = new outerclass();
create inner obj ← outerclass.innerclass obj2 = obj1. new innerclass();
```

$\underbrace{\text{outerclass.innerclass}}_{\text{class name}}$ obj2 → inner obj; obj1 → outer obj; innerclass() → innerclass.

```
    obj1 → outer obj
    obj2 → inner  "
  //call fun using inner obj
    obj2.innerfun();

  }
} // demo ends.
```

Static Class

- An innerclass can be declared as static
- The purpose of static innerclass is that its obj can be created <u>w/o creating obj of outerclass</u>.

Note: outer class can't be st.
 Only inner class can be static.

```
outerClass {

        PV outerfun() {
              //define
        }

        Class static innerclass {
                        PV innerfun() {
                                //define
                        }
        }

}
```

CD { PSVM (---) } → Class demo {
Pub st. void main(..)}

//No need to create obj of outerclass.
// obj of innerclass can be created directly

$\underbrace{\text{outerclass.innerclass}}_{\text{Class name}}$ obj = new $\underbrace{\text{outerclass.innerclass}}_{\text{Class name}}$();

obj ↓ innerclass obj

*** //Here innerclass is static, so its obj can be created directly w/o creating obj of outerclass.

Import & Static Import

(Unit-2)

- We can import any class & use it.

eg ~~jat~~ import java.lang.System;

```
CD { PSVM() {
        System.out.println();
        ↓
     use System class after importing
     }
}
```

Note :- System class is auto imported, but it can be manually imported too.

What does System.out.println() mean

```
class System {
Static PrintStream out = new PrintStream();

}
```

```
//System is a class
//out is an obj inside System.
//out " not an obj of System, but out is an
    obj of PrintStream class.
//out is static, so, it can be accessed by using
    class name
        System.out...
```

What is static import

- Another way to implement a class through <u>static import</u>

How to statically import

1) ~~use~~ use static KW
   import static java.lang.System.*;
2) when statically importing, use * after class name wh. means all members of a class are imported & not just a particular class.

What happens when u statically import a class? Benefit?

- We can access members w/o using <u>class name</u>

eg System.out.println(); //Normal import
out.println(); //with static import,
<u>class name</u> is not needed.

- So, it reduces effort of typing classname everytime.

import vs Static import

import	Static import
• Syntax: & import <class>	import static <class>.*
eg import java.lang.System;	eg:- import ~~#~~ static java.lang.System.*; * means all members of class.
• class name must be used System.out.println();	• members can be accessed w/o class name out.println();

Note :-

- St. import doesn't make a class static or it doesn't make the members static.
- The class members r already static.
  - It simply allows to use the members directly w/o using class name.