1. Good Evening
2. We will begin at 9:08 pm
3. Topic - OOPs 2

Agenda

1. Objects in memory] Types < val / ref
   - Fns call
     - call by val
     - call by ref
2. Constructors
   - Default
   - Parameterize / Manual
   - Copy — Shallow
     - Deep
3. Destructors
4. Access Modifiers → Public, Private
   Default / Package

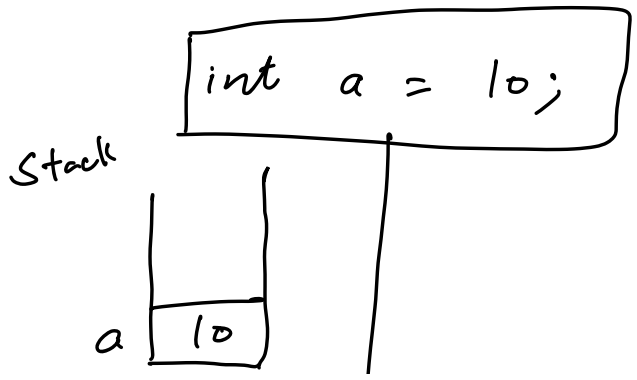
Objects in memory

Two kind of datatypes

1. Primitive or Value type
2. Referrence Types
3. Special → String

Primitives : 8 primitives

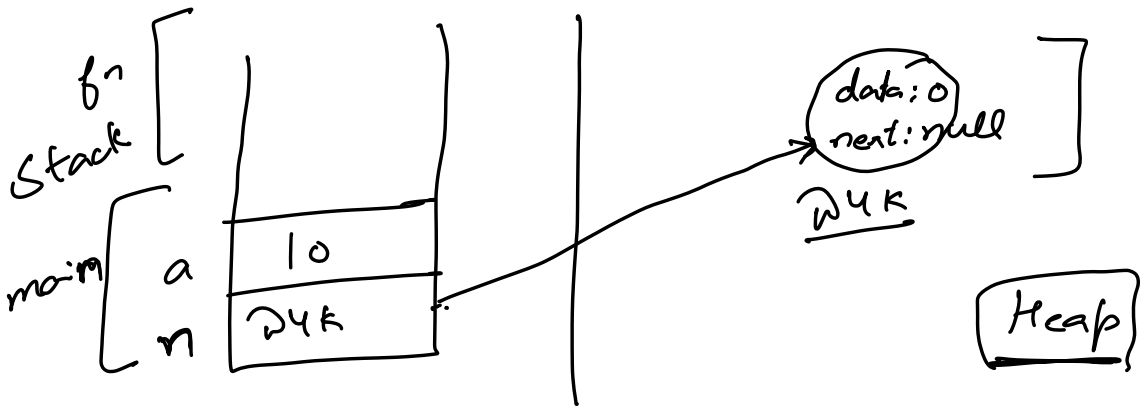
- byte, short, int, long → 0 is default value
- char → '\0'
- float, double → 0.0
- boolean → false



☆ Value type are stored in stack.

Reference Type → Anything besides primitive

```
class Node {
    int data;
    Node next;
}

Node n = new Node();
```



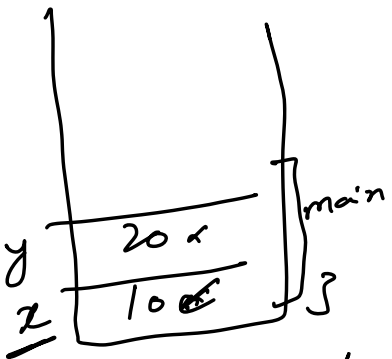
All ref types have a default value of null

Value	vs	Ref type
→ Stored in stack		→ Stored in heap & addresses stored on stack
→ Different default values for each primitive		→ Default value is null

---

```
main () {
    int x = 10 ;
    int y = 20 ;
    fun (x, y) ;
```

→ [x & y] → 10 & 20




```
fun ( int xx , int yy ) {
    int temp = xx ;
    xx = yy ;
    yy = temp ;
```

→ [xx & yy] (20, 10)

```
}
```

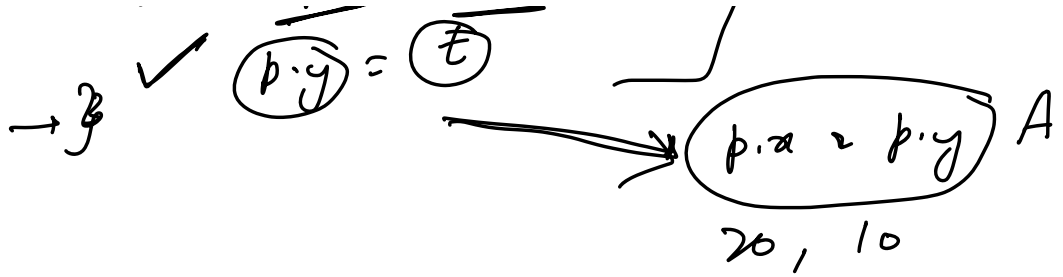
Val types called by value

Case 1 → Is it clear?

[Type → value type
  Call → By val]

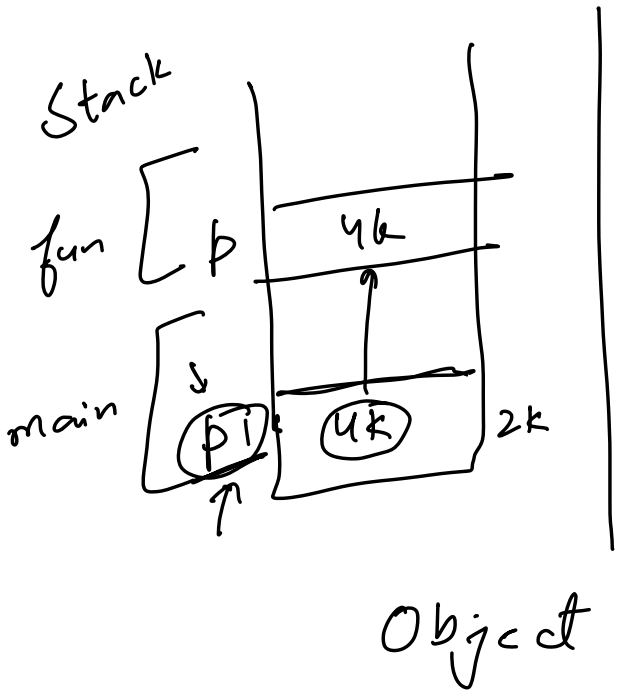
Case 2

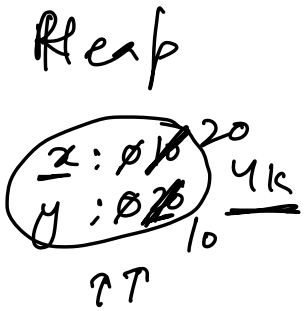
```
main ( ) {                          class Point {
✓ Point p1 = new Point()                int x;
✓ p1.x = 10                             int y;
✓ p1.y = 20
→ fun (p1) ←                        }
}                                   B (p1.x & p1.y)

fun ( Point p ) {
✓ int t = p.x ✓
✓ p.x = p.y
```



A : $p_A.x$, $p_A.y$ 20, 10

B : $p_1.x$, $p_1.y$ 20, 10





Ref → Instance

Object

Example → Type → Ref type

Call → value

→ Val type . [called by value]

- Ref type, called by value

Request → This may seem basic, but it is very fundamental

---

main ( ) {

✓ Point p1 = new Point ( )

✓ p1. x = 10

✓ Point p2 = new Point ( )

✓ p2 . x = 20

✓ fun (p1, p2)

B → p1.x, p2.x

10, 20

}

fun (Point p11, Point p12) {

✓ [Point temp = p11;]

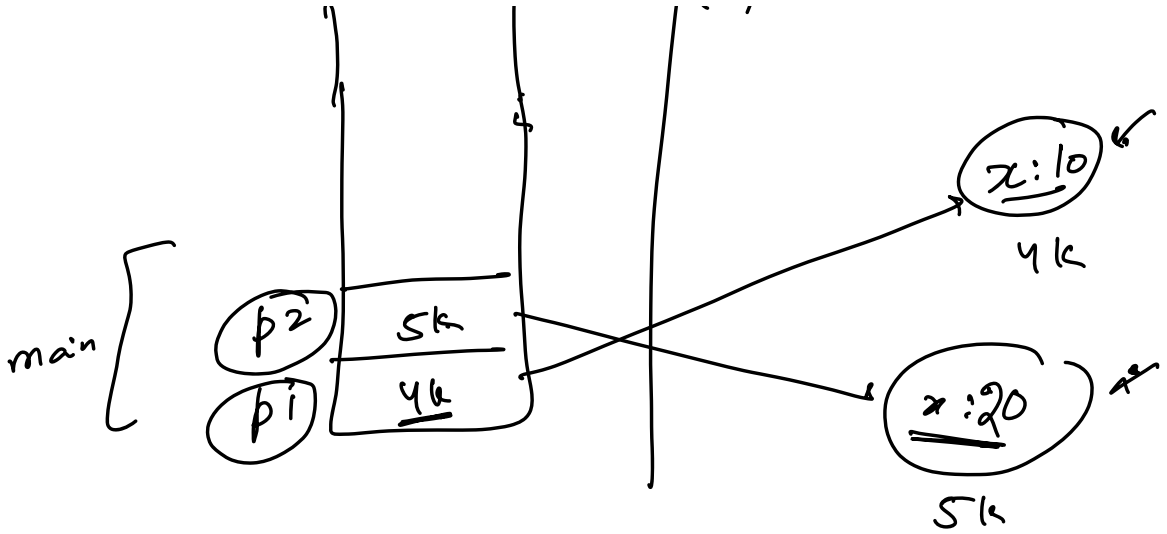
✓ p11 = p12

✓ p12 = temp

A: → p11.x, p12.x

20, 10

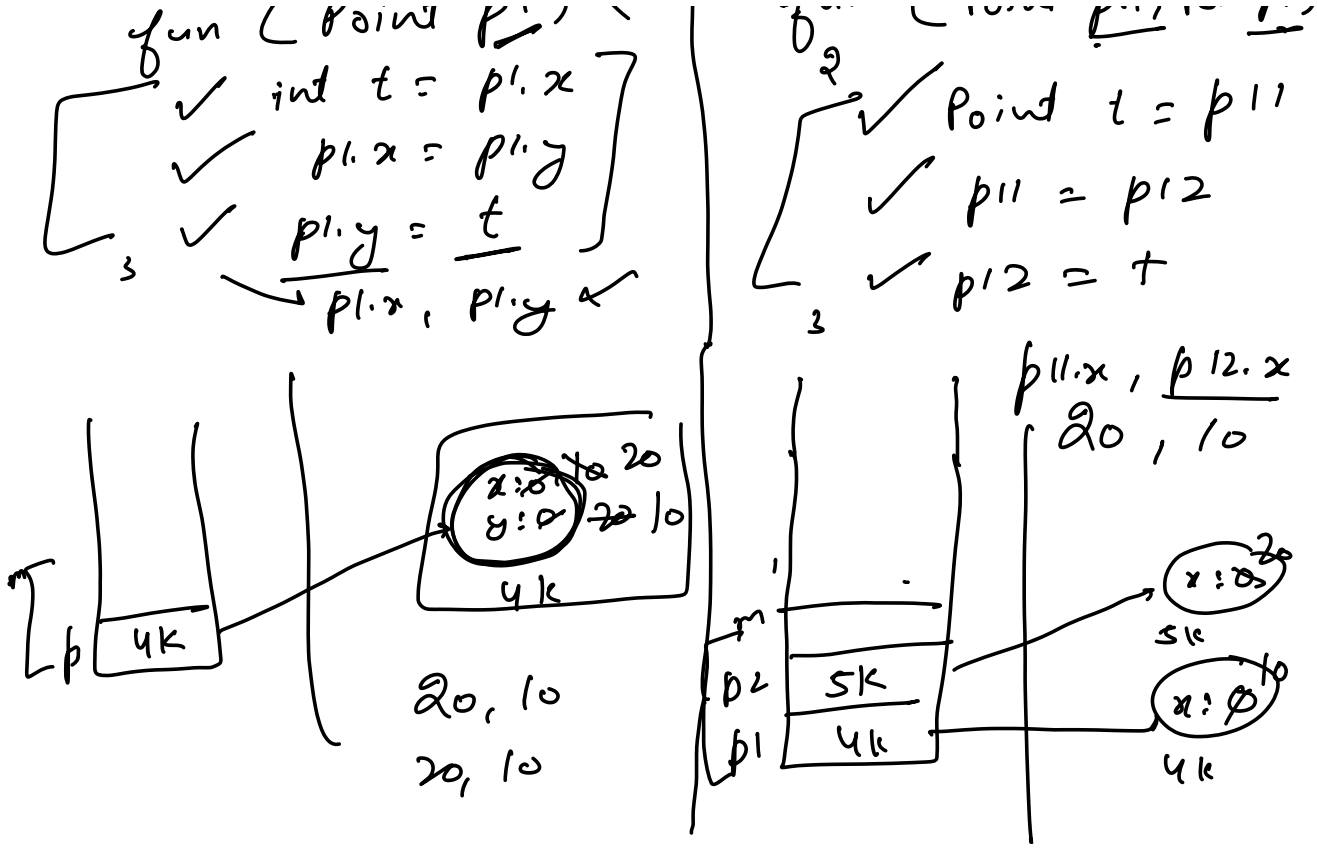
}



8 : 12
× ✓

main ( ) {
✓ Point p = new Point()
✓ p.x = 10
✓ p.y = 20
✓ fun (p)
→ p.x, p.y
}

main ( ) {
✓ Point p1 = new Point()
✓ p1.x = 10
✓ Point p2 = new Point()
✓ p2.x = 20
✓ fun (p1, p2)
→ p1.x, p2.x
(0, 20
}
fun (Point p1, Point p2)



Break → 10:19 to 10:28

- String , StringBuilder
- Cfor
- Dtor
- AM

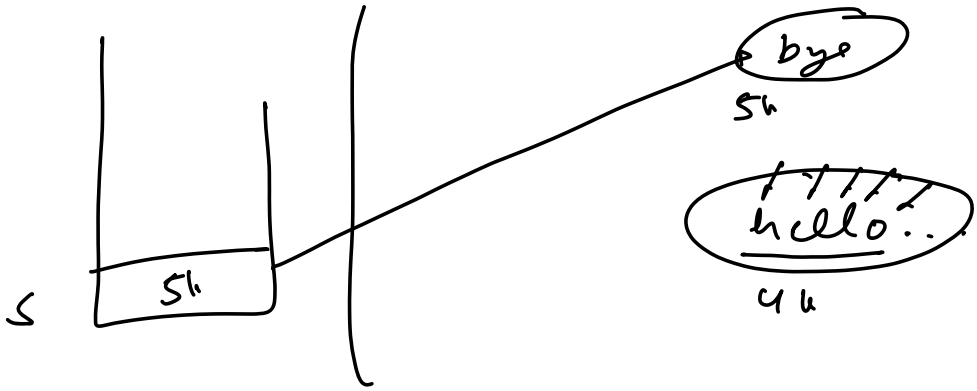
String

└ A special ref type which behave like value types.

→ Immutable

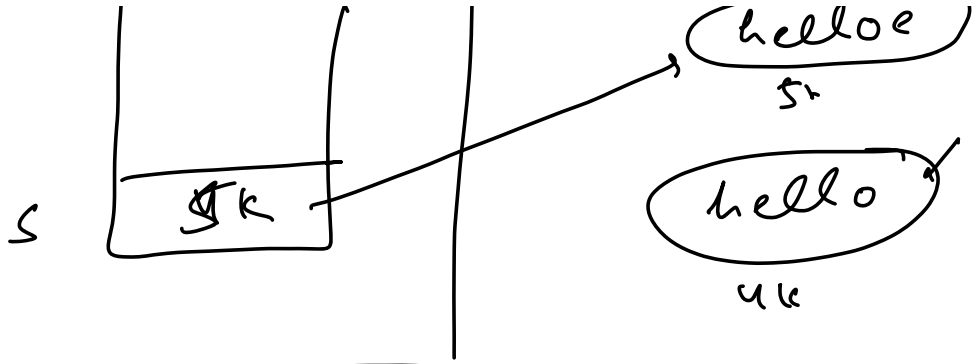
✓ String s = "hello";

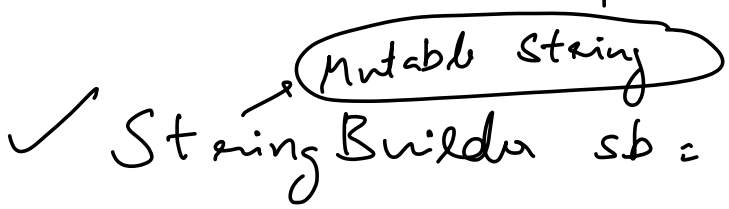
✓ s = "bye";



✓ String s = "hello" ← 2 < B

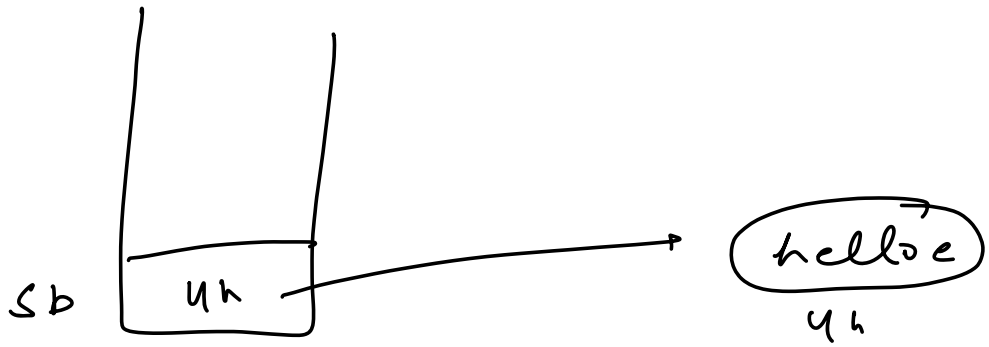
✓ s += "e"; slow





✓ StringBuilder sb = new StrB("hello")

✓ sb.append('e');



C++ String ≠ Java String

= Java StringBuilder

String, SB, SB better?

Constructors

```
main() {
    Point p = new Point();
    p.x = 10;
    p.y = 20;
}
```

```
class Point {
    int x;
    int y;
}
```

What is a ctor?

- A special fn. called during object creation
- It has no return type
- It has the same name as class
- Used with new keyword
- Assigns values to data members of new object that has been created

Default ctor

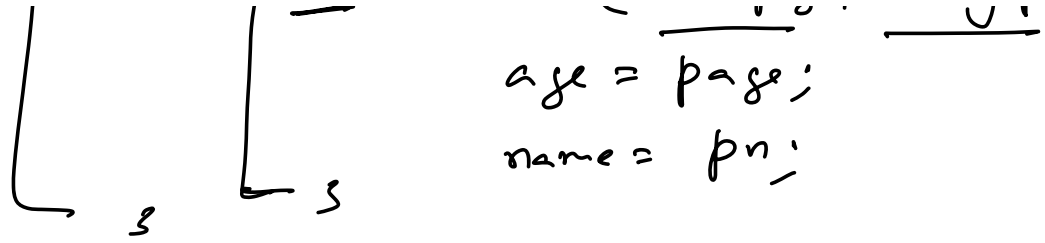
Default Ctor

- If we donot add any ctor to a class, Java automatically adds a default ctor to the class.
- This default ctor assigns default values to data members

```
class Person {
    int age;
    String name;
}
```

Parameterized Ctor

```
class Person {
    int age;
    String name;

    Person (int page, String pn) {
```



```
    age = page;
    name = pn;
}
```

```
main() {
    Person p = new Person();
    p.age = 10
    p.name = "Aryan";
}
```

```
main() {
    Person p = new Person(10, "Aryan")
}
```

- Parameterized
  - Added by developers
- Default
  - Only when there are no other ctors, Java adds a default ctor

COPY Constructor

↳ To create object from another object

```
class Point {
    int x
    int y;

    Point (int xx, int yy) {
        x = xx
        y = yy
    }

    Point (Point p) {
        x = p.x
        y = p.y
    }
}
```

```
main() {
    Point p1 = new Point (10, 20);
    Point p2 = p1
    p2.x = 100
    Point p3 = new Point (p1);
    p3.x = 1000;
}
```

Default, Parametrized, Copy

* Destructors → A fn. which is run to reclaim resources [memory] from object before it is destroyed.
  - C++
    - Don't have Garbage Collector
  - Java has GC

main { }

In C++ if we make an array on heap
  - delete keyword

Person ? | class Person ?

File f;

void finalize() {
    f.close();
    f.dispose();
}

~Person() {
}

To free up other resources like files, sockets

→ Garbage Collectors

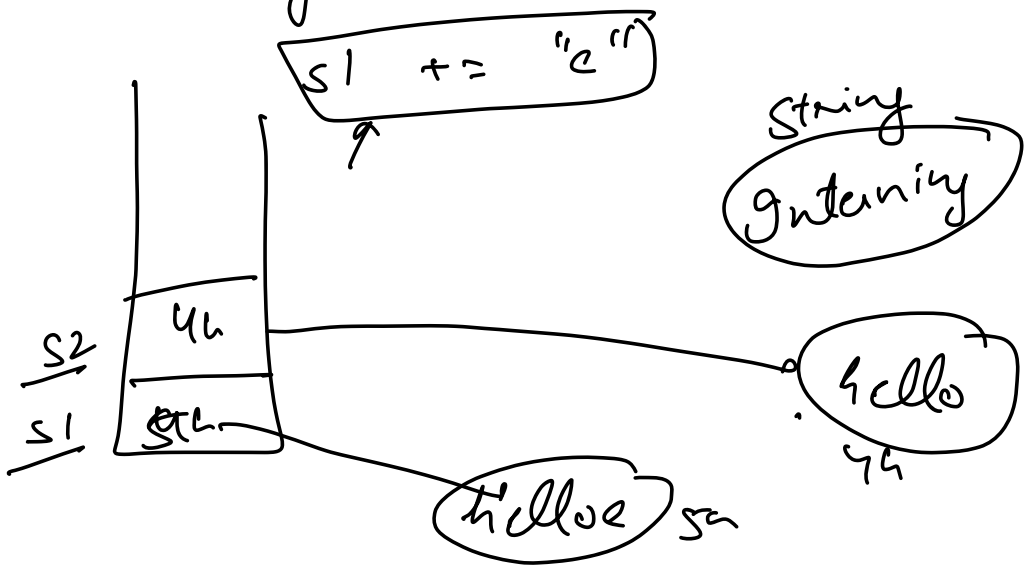
→ finalize fn.

Access Modifier →

Immutable?

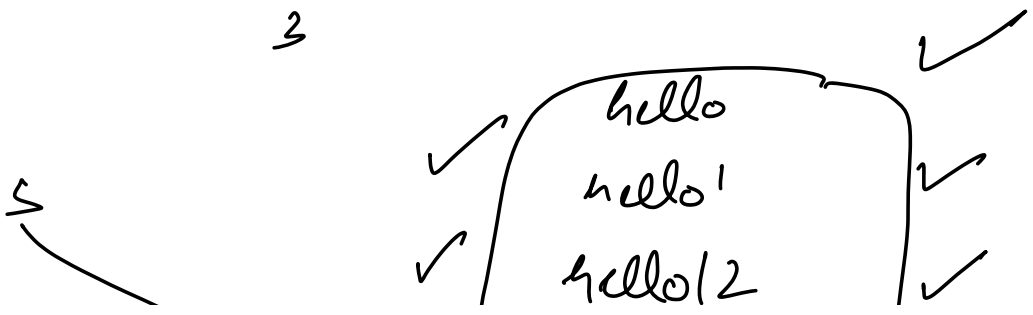
String s1 = "hello"

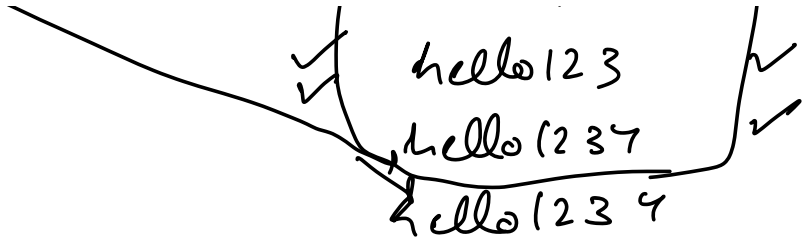
String s2 = "hello"



gntening

```
String s = "hello"
for (int i = 1; i <= 5; i++)
    s += i
```

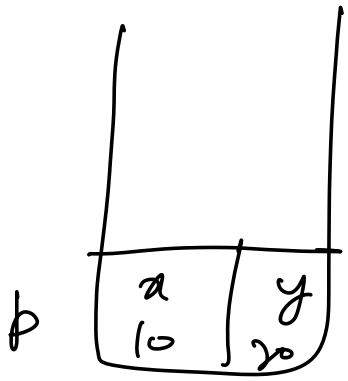




Access Modifiers

```
struct Point {
    int x;
    int y;
};
```

Point p = 3




```
{ Person p1 = new Person(10, "Aryan")
```

```
{ Person p = new Person();
    p.age = 10
```

p. name = "Arjan";

Abstract, Inheritance