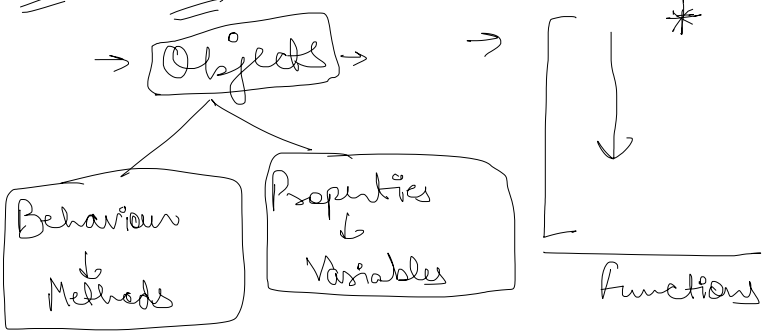
Doubts

1. Finalize (Garbage Collection) VS Final vs Finally.
2. Synchronized Methods (Multi-Threading) → (wait && sleep)
3. Static Keyword *
4. Abstraction vs Encapsulation
5. .toString()
6. (==) vs equals()
7. Overloading vs Overriding (Run vs Compile) (Static vs Dynamic Binding) [Polymorphism]
8. Friend Functions
9. JDBC
10. String Buffer vs Builder
11. Classloader
12. Copy Constructor vs Virtual Function
13. Serialization
14. Virtualization.
15. Composition vs Aggregation vs Association.
16. Linear vs Non-linear D.S.
17. Hierarchy of Collections
18. Linked list vs Arraylist
19. This Key Word.
20. Run vs Start
21. Process vs Thread
22. Heap vs Stack Memory.
23. Generics
24. Comparator Interface. vs Comparable
25. Abstract Keyword → Interfaces Also
26. Type Casting → 2
27. Scanner vs Buffred Reader] File Handling
28. Exception → Inheritance
29. CountDownLatch *
30. Tokenizer String
31. Why [OOPS vs Psoudural]
32. Bitwise operators
33. → What Platform IOS is build upon? [(?]
34. Char to int is Casting?
35. Access Specifies
36. Super in Interfaces

1. OOP VS Procedural *

1. OOP vs Procedural



Divided into Objects
Bottom-Up Approach

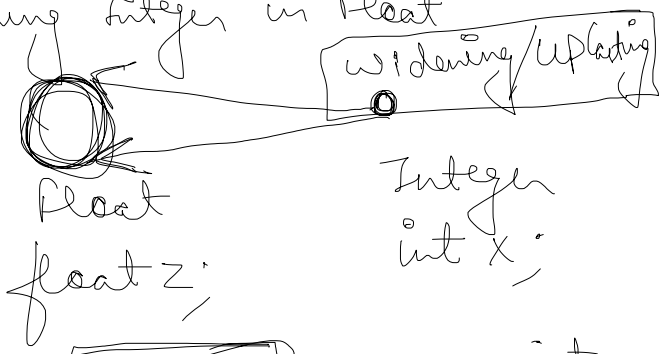
Top-Down Approach
Not Easy to add
No Polymorph

Purely Real World

X Real World

2. Type-Casting
   - converting one value from one type to other

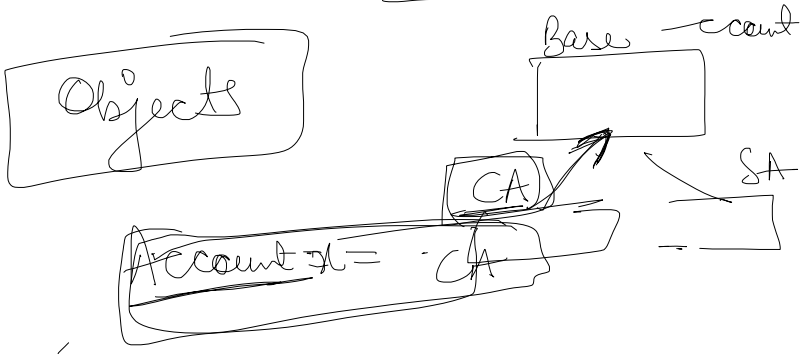
1. Storing Integer in float



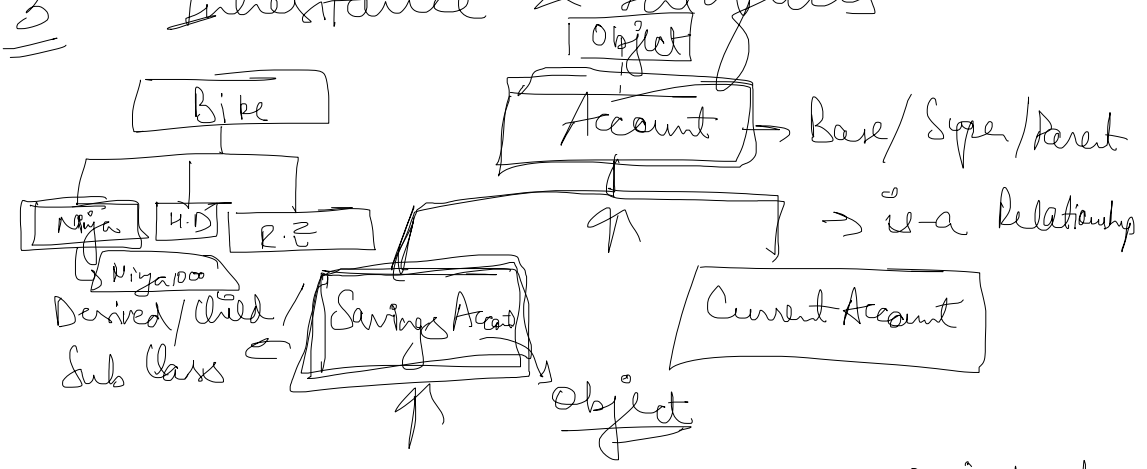
2. Storing float in int

Narrowing/Down Casting

float z
int x
x = (int) z
Explicit



3 Inheritance & Interfaces



Object of Derived Class - is an Object of Parent class.

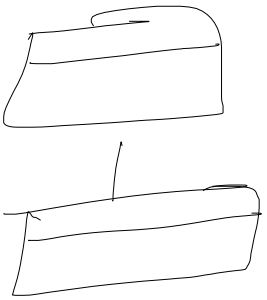
SavingsAccount x = new SavingsAccount();

extends Keyword Inheritance

Syntax

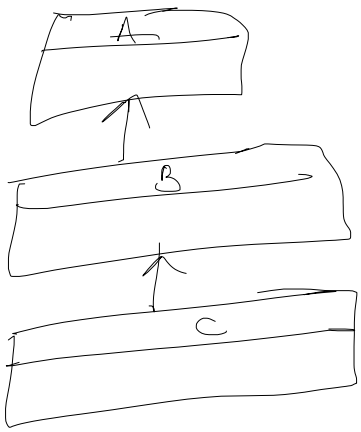
Class Derived extends Base

1

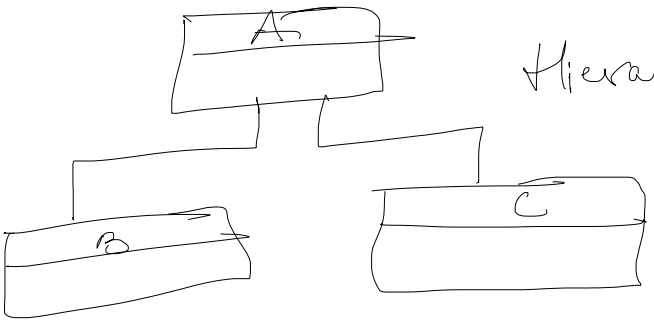


Single

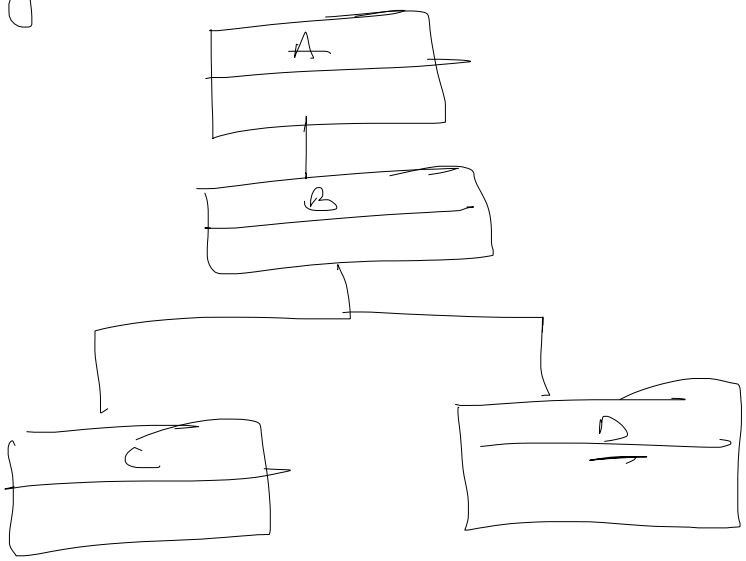
2 Multi-level



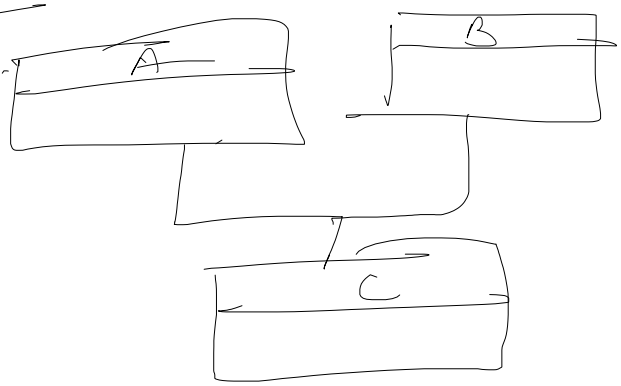
3 Hierarchical



4 Hybrid



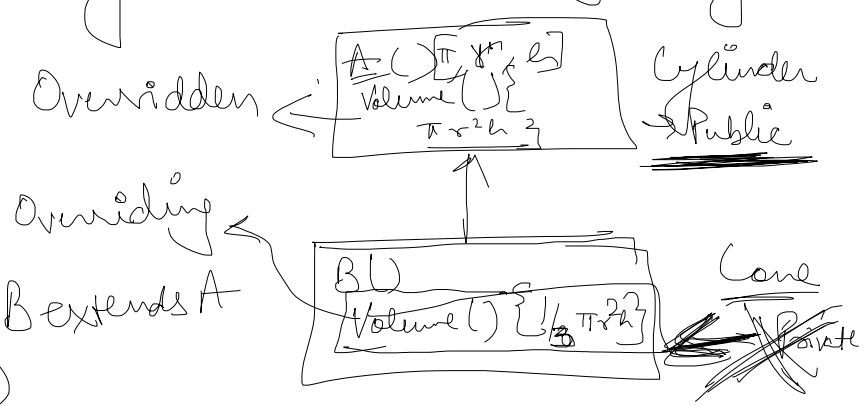
5 Multiple



Not through classes.
Only through interfaces

```
class A
{ void Test()
  { S.O.P("Hi I'm A");
  }
}
class B
{ void Test()
  { S.O.P("Hi I'm B");
  }
}
class C extends A extends B
{
  S.O.P(Test());
}
```

4 Overriding && Overloading / Polymorphism / Binding



c()
↓
baby.volume()

π, r, h / Volume

B. value → Long

Overriding → .Equals() [String]
Content of Object.

→ Signature of Overridding func should be same as that of Overridden function.

→ Access Modifier of overridding method must have higher/equal level of accessibility

Base → public int fun(int i, float f, String s)

Yes ← public int fun(int i, float f, String s)
No ← protected int fun(int i, float f, String s)
No ← public float fun(int i, float f, String s)
No ← public int fun(int i, float f)
No ← public int fun(float f, int i, String s)
No ← public int fun(int f, float s, String f)

Method Overloading

Within the same class

Salary Calculator
{ Calculate(int Salary)
{ return Salary * 12

Method

Method Overloading

```
return Salary * 12
}
Calculate(int Salary, int months)
{
return Salary * months;
}
```

3

→ No. of Parameters
→ Types of Parameters
→ Return types not considered.

```
class Test
{
void fun(int, int)
void fun(int, char)
int fun(int, int)
}
```

Error

Confusion

```
class Main
{ PSVM(S A[]){
fun(5,5);
fun(5,a);
fun(5,5);
}
```

3

```
class A { void Test() { } }
class B extends A { void Test() { } }
class C { main() { B.Test(); } }
```

Run Time

Method Overloading ← println()
println(int)

Overloading → Println(int)
(float)
(float, int)
etc... etc.

Poly Morphism
↳ ways to Exist
Multiple

KeyWords

Super → Refers to the Parent/Base class of class in which it is used.

A
int a = 5
fun()
S.o.p("in A")

B
int pi = 7
S.o.P(pi);
S.o.P(supp i);
fun(S.o.p("in B"))

1.
2.
3.

fun(); → B
superfun(); → A

Variables (Instance)
Methods
Constructor

Overriding

Default
Super();
Super(parameters)

Derived

Constructor Overloading

class A
{ int n;
A() {n=5}
}

Parent first

class B
{ B() {S.o.p(n);} → Exception
}

B.obj = new B();

This Keyword

This Keyword

It is used to refer to the current instance of a class

```
class Students {
    String name;
    int roll;
    String college = "VIT";
    Students() {
        this.name = null;
        roll no = null
    }
    Student(String name, int roll)
    {
        this.name = name;
        this.roll = roll;
    }
}
class Main { PSVM(S A[])
    {
        Student A = new Student();
        Student B = new Student
            ("Sehajpreet", 0345);
    }
}
```

Page - 76

Abstract Key Word

↳ 1 Method
2 Classes

abstract Methods → Declare but no implementation

```
abstract void fun();
```

Abstract void fun();

final ~~static~~

→ Any class having an abstract method will become abstract class

Error ← class Test {
abstract void fun();
}

abstract class Car {
abstract void accelerate();
abstract void brake();
}

abstract class Audi extends Car {
accelerate { -- }
}

class AstonMartin extends Car {
void accelerate() { S.O.P(" ") }
void brake() { S.O.P(" "); }
}

Abstract class cannot be instantiated. [Cannot be used]

~~Audi A8 = new Audi();~~
~~AstonMartin~~ m300 = new AstonMartin();

→ Abstract methods are used to force the subclasses to provide their implementation by defining the body.

→ Abstract class can also have concrete methods [Means, abstract class does not necessarily have only abstract methods]

→ A class without any abstract method

can also be declared as abstract using the keyword.

→ A class with an abstract method will be abstract for sure

→ It is not necessary to override the concrete methods of an abstract class.

```
abstract class hello
{   abstract void demo();
    abstract void fun();
    void printing()
    {S.o.P("Hi there");
    }
}
```

Instantiate ✓

```
class check1 extends hello
{   void fun()
    {
        S.o.P("overiden fun");
    }
}
```

```
abstract class check2 extends hello
{   void fun()
    {
        S.o.P("overiden Print");
    }
}
```

✓ class temp extend check2 { void dem }

```
class check3 extends hello
{   void fun()
    {
        S.o.P("overiden fun");
    }
}
```

```
void printing ()
{
  S.o.P("overridden print");
}
```

}

Final Keyword

No relation with each other

- → Method → Cannot be overridden
- → Class → Cannot be inherited
- → Variable → Value once assigned cannot be changed

```
final int i = 5;
```

```
class demo{
  final int i;
  demo (int i)
  {this.i = i
  }
}
```