* Up Casting :

Upcasting is a type of object type casting in which a child object is typecasted to a parent class object. By using the upcasting, we can easily access the variables and methods of the parent class to the child class. Here, we dont access all the variables and the method. We can access only those method of child class which are in parent class and it will print the overriden method.

example:

```
Class Employee {

}

class FTE extends Employee{

}

Class PTE extends Employee {

}

Employee emp = new FTE ( );
```

Date.
Page No.

→ We are able to access those features which are coming from the parent class and overridden in a child class.

→ We are not able to access those features which are specific to the child.

Upcasting → Narrowing →

Example:

[Hanuman losing power story]

Hanuman is a super human and he was upcasted to its parent class human by Rishi.

2* Down Casting

For downcasting we need to check the types or else we may get Class Cast Exception.

→ Downcasting can be considered as the reverse of Upcasting. We are type casting an object of the parent class to the child class.

→ instanceof operator :- This operator returns true if the object is of a particular type.

→ Cast(), isInstance() method can replace the instanceof operator.

Example:- if(emp instanceof FTE)

FTE fte = (FTE) emp;

* To restrict ambugity in parent class, we use abstract class *

*** Abstract Class** Abstract class act as a most generic class and it give the features to its child class.
We restrict the object creation of abstract class because in nature it is too generic.
eg: Account, Loan, Product, etc.

Que: Do abstract class has constructor?

Ans Yes, because constructor are initialized by super keyword from the child class.

→ Abstract class may or may not have abstract methods.

→ If we define abstract methods, so it is must to ~~have~~ ~~the~~ make the class abstract and the child class must override abstract method otherwise child class become abstract and we cannot create its object.

Date.
Page No.

* Partial Implementation of Abstract class :-

→ If we have abstract class (Parent), it has 30 abstract methods.
And our child will override 10 out of this, so our child is also becomes abstract and we have a grand child which create or implement set of 20 methods, so it complete the abstract methods.

→ Abstract method do not specify a body.

* ~~Inheritance~~ Interface i.e 100% abstraction :-

(*) Modes of Interface :

(1) Standarization :- It sets standard / Prototype.

CRUD → [Create, Read, Update, Delete]

*→ Interface ~~know~~ is used to declare what to do.

→ Interface does not ~~have~~ know how to do, this is the job of classes.

Example:

```
interface Player {.   → Abstract interface Player {
                        Default
    int MAX_JUMP = 100;   // public static final int max_JUMP = 100
    void jump();          // public ~~static~~ void jump
    void Kick();          → abstract
}
```

→ In interface, we donot have to mention that which method is abstract because by default all methods are abstract in interface.

→ Also a class can inherit more than one interface which is not possible in case of class and abstract class.

↳ i.e multiple inheritance using interface.

→ There is no problem of ambiguity in multiple inheritance using interface because in interface all the methods are abstract and there is no logic inside abstract method.

→ When we use static then variable or method are bind with class.

i.e
```
class Parent {
int var = 100;
void main() {
}
}
```

Parent. var;

Parent. main();

→ When we create object i.e using new then methods and variables are bind with object.

Example:-

Student st = new Student();

st. main(); , st. age,

Date.
Page No.

→ There is no case of ambiguity in case of instance variable in multiple inheritance using interface in java because, by default all the instance variables are final in interface. So we can access them using class name to avoid ambiguity.

x x x x